DESIGN OF A MICROPROCESSOR CONTROLLED FIVE AXIS NC MACHINE

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

M. KRISHNASWAMY

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

FEBRUARY, 1986

21/2/8

CERTIFICATE

It is certified that this work entitled 'Design of A Microprocessor Controlled Five Axis NC Machine', by M. Krishnaswamy has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

Dr. S.K. Mullick
Professor
Dept of Electrical Engg.
I.I.T. Kanpur.

Dr. S.G. Dhande
Professor
Dept of Mechanical Engg. and Computer Science
I.I.T. Kanpur.

ACKNOWLEDGEMENTS

It is with great pleasure that I express my gratitude to my thesis supervisors Dr. R.M.K. Sinha, Dr. S.G. Dhande and Dr. S.K. Mullick for their guidance and constant encouragement.

I am grateful to the timely advises and help of Dr. R. Raghuram, Dr. M.M. Hasan and Dr. P.K. Chatterjee.

I thank Dr. K.R. Srivathsan and Dr. R.N. Biswas who taught me microprocessors and electronic circuits.

Once again my thanks are due to Dr. S.G. Dhande who gave m the topic and whose guidance and encouragement made this thesis materialize.

My thanks are also due to my friends Philips, Nithy, Abey, Sukumar, Ponnu, Sampath and also to ACES Staff who helped me during my stay at IIT Kanpur.

I wish to thank Mr. C.M. Abraham for his fast and good typing.

M. Krishnaswamy

ABSTRACT

A microprocessor controller of a five axis NC machine with three simultaneously controlled axes has been designed. The system accepts a set of G-code instructions and drives the machine accordingly. The software and hardware have been implemented and tested successfully; the machine being simulated by a plotter. The plotter has been used in the incremental mode. The hardware consists of a SDK-86 kit, based on the intel 8086 16 bit microprocessor, a CRT with keyboard and the plotter. The software has been written in the PLM-86 language and developed in the MDS series III system and implemented as firmware in EPROM's. Some illustrative examples showing how the system can be used are also given in the present work.

CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Classification of NC Systems

After the introduction of NC machines which consisted only of hardware and accepted the punched paper tape as input, the mechanical automation was increasingly replaced by computer automation in NC machines. This led to the invention of a wide variety of NC systems. As these new types of machines were widely used, more tasks arose which encouraged the adoption of small or large computers to take full advantage of the possibilities offered by the concept of numerical control.

Initially computers were used off-line to compute the interpolations, offsets etc. But, as microelectronics developed fast, NC machines became available with a special-purpose computer to compute for a group of several machines. Nowadays, with the advent of microprocessors manufacturers are able to integrate the computing functions within the NC machines. As computers became common place, high level part programming languages and compilers were available. Now, there are NC machines which have the part programming interpreter built into the machine itself, which eliminate the need of using a mainframe computer for less complex computations.

Initially, the large installation and maintenance costs of the computers forced the users to control many NC machines through one large computer. This system is called as direct numerical control (DNC) [2,3].

Controlling one NC machine through a special purpose computer is called Computer Numerical Control (CNC) [2,3]. The CNC needs just enough memory and I/O ports needed to store utility programs and user programs and to communicate with the machine tool. Hence the cost is less compared to DNC systems. Though small in its class, the micro-computer in a CNC system has enough memory storage eliminating the need for the punched tape, and providing some editing features as well as a cathode ray terminal (CRT). More programs for diagnostics, debugging, self-test etc., are also implemented in the CNC machine.

When the special purpose computer used in the CNC system is replaced by a microprocessor system, as it is being increasingly done, it becomes a micro-computer numerical control system. This system offers the advantages of low cost, reliability, low maintenance costs, and small space [1].

CNC and DNC are really forms of computer aided manufacturing (CAM). More frequently the system is also used for collecting management information like, the number of parts made, down time, set up time and types of machine failures. The

same system can be used for scheduling and inventory control.

When a computer is used to design the parts, with the help of an interactive graphics hardware and software, the process is called computer aided design (CAD) [2]. The design then will lead to automatic simulation, analysis and evaluation. This computer can then deliver the design to the CNC machine in some machine readable form, with the help of a part programmer. In such a case, the limitations imposed by the small memory storage and computational capability of CNC or micro-computer NC machines can be overcome with the addition of a communication link with a CAD environment. This link can be a simple RS-232C link or current loop or a telephone line with a modem at each end.

Such a link increases the possibilities of complex machining operations. Also, if the mainframe computer is networked with other computers, the design by a CAD environment elsewhere can be used by the local CNC machine easily. Use can also be made of the mass storage media of the mainframe computer, to store the developed part programs.

1.2 Development of Microprocessor Systems for NC

With the advent of microprocessors the computing systems for the NC machines are being implemented with fewer chips, with fewer interconnections and hence with higher reliability. The cost versus performance ratio of microprocessors has been

on the decrease ever since the introduction of the first generation of microprocessors, i.e., intel 8008, in the year 1971 [1] concurrently the memory chips, peripheral chips and other logic chips have been improved in performance, reliability and cost. Now 16-bit and even 32-bit microprocessors are available so that the computations which were earlier done in a large computer can now be done in the CNC machine itself.

Present day microprocessors fulfil all the requirements of the CNC machine, like speed, memory capacity and computational capability. Even the generation of the timing signals and the feed frequency can be realised through the software in the μP system. If more computational time is needed for arithmetic operations, it is possible to use one or two peripheral chips to do the same job of controlling the drives. The development of the system is based on the considerations like the maximum cutting speed, the number of simultaneously controlled axes, the interpolation accuracy needed and the interaction between the user and computer. Based on these, a compromise is struck between software and hardware to optimise speed and cost.

The μP system for NC is designed after finalising the hardware-software compromise. The hardware consists of the CPU, control chips (if needed), memory, Input/Output chips and the drive interfaces. The basic hardware can be tested with the

help of a logic analyzer. The software is developed concurrently in a micro-computer development system (MDS). The bugs can be removed using the debugging package in the MDS. Then the software is written into or 'burnt' into EPROM memory chips and can be installed in the hardware. The MDS has an in-circuit-emulator (ICE) which is used to debug the hardware and software simultaneously. After the hardware and software are ready, test data can be input into the microcomputer and the system performance evaluated.

1.3 Point-to-Point and Contour Programming

There are two types of numerical control, each used for a different kind of machine operation. One is called point-to-point, or positioning control, and the other is called contouring, or continuous path numerical control [2,3].

In the point-to-point numerical control the cutting tool or the work table is moved to a specific position as called for on the part program and an operation like drilling of a hole is performed. A simple drilling machine follows this type of control. The exact path taken by the tool is immaterial, and only care should be taken such that the spindle does not collide with either the part or the holding fixtures.

Unlike the point-to-point control, contouring control requires that two or more motions of the machine be simultaneously and precisely controlled. In the contour control, each

incremental movement must be described together with its feed rate number. The entire travel must therefore be controlled to close accuracy both as to position and velocity. Milling machines are proven to be the popular application of continuous path numerical control. Most contouring machines move only in straightline movements and the circular and other cuts will be broken into short straight line segments. The breakup of a curve into these straight line segments is one of the major time consuming requirements in contour programming. Another calculation required is describing the path of the center of the cutter rather than the actual part contour dimensions. This is called the offset calculation. The problem becomes complicated as the number of simultaneously controlled axes increases.

The contouring machine has at least the linear interpolation capability. The interpolation is done either in the hardware or through software. The velocity along the cutting path should be constant irrespective of the direction of travel. The straight line is split into a number of small segments and each segment is travelled in a fixed time interval, so that the timing frequency determines the velocity of the tool cutting. The timing frequency is varied according to the feed rate required.

1.4 Objectives and Scope

In this thesis, design of a microprocessor based five axis NC machine is described. The experimental set up and the program controlling it are also described. The objective of the work is to develop a base for a microcomputer controlled machining center. Though the design is now for only G-code input, the design is done with the view of future work, like implementation of a part programming language compiler/ interpretter with more interactive programming. The experimental set-up takes the G-code sequences as the input through the CRT and processes them to generate the control signals and information required by the drive system to move the tool in the required path. The set-up also generates additional signals like coolant on/off signal, spindle speed change information, spindle direction signal, annunciator signals, etc. The set-up senses various switches like manual positioning switch, emergency stop switch, hold switch, program reset switch etc., and takes appropriate action. A plotter version of the set up which controls the movement of a pen in a two axis plotter is also described.

CHAPTER 2

DESIGN OF THE μP CONTROLLER

2.1 Description of experimental set up

The microprocessor chosen for this particular application is the 16 bit intel 8086. The selection was based on the following considerations :

The system should be capable of being modified in future to have more options and high level part programming language implementation.

It should be fast enough to do real time interpolations like for circles and complex curves in future.

The system should be designed and the software, hardware debugging should be done locally. Because of the availability of the microcomputer development system for the intel products, intel 8086 [4] was chosen to be used as the central processing unit.

The availability of the high level language PLM encouraged all the software be written in a highly readable form. The SDK-86 Kit [5] was used in place of the basic microcomputer hardware.

The RS232-C interface using the 8251 SIO chip is used in the set up to the data entry and display functions through a CRT monitor. The program is input through the keyboard with the help of a small editing facility provided in the software. After all the data are entered, the CRT displays the final version of the G-code sequence and waits for a go-ahead signal for machining, from the operator. When it is given, the machine starts moving the table according to the instructions in the G-code sequence.

Provisions in the software allow a paper tape reader to load the G-code instructions direct to the memory from a paper tape.

The intel 8253 timer/counter chips [4] are used to generate the feed frequencies for stepper motors which drive the table and also to count the number of pulses fed to the stepper motors. Each of the two 8253 chips contain three programmable timer/counters. Each of these can be programmed in a wide variety of modes.

In this application two such chips are used, one for generation of the rate signals for the stepper motors which control the velocity in each direction and the other is used for counting the steps in each direction and stopping the pulses fed to the stepper motors when the correct number of steps are done. The first chip's three timers are programmed in mode 3 which

is a square wave generator mode. Each of the 16 bit timers are used for a separate axis (X,Y or Z). The second chip's timers are used in the mode Ø, that is, interrupt on terminal count mode. The data corresponding to the number of steps to be taken in each direction is programmed into each of the three 16 bit counters.

The principle of operation is described as follows :

When it is required to move the tool from (X1, Y1, Z1) to (X2, Y2, Z2) at a particular feed rate F, the component rates at each of the X,Y,Z directions are different depending upon the values of (x2-x1), (y2-y1) and (z2-z1) and F. These velocity components are calculated and accordingly the rate generators (i.e., the timers 1,2 and 3) are programmed to these velocities as accurately as possible. Any inaccuracy in calculating the velocities should not offset the end arrival point. For this, the steps in each direction are counted and when all the steps in one direction are done, the pulses to that stepper motor are blocked by the output of the corresponding counter 1,2 or 3. To get this 3-D linear interpolation practically accurate, it is often necessary to break the long straight line into small segments and programming the counters for these segments. This way the tool travel path is kept almost linear.

The block diagram for the experimental set-up is shown in Fig. 2.1. The detailed schematic for the timer interface is shown in Fig. 2.2.

To generate the various auxiliary function signals and to read the various manual switches the 8255A Programmable Peripheral Interface (PPI) [4] chip is used. Each chip consists of three eight-bit ports which are software programmable to be configured as inputs or outputs or bidirectional ports separately. For the purpose of demonstration, simple LED indicators have been used to show the levels of the output port's signals which correspond to the auxiliary functions. Also the direction signals for the stepper motors are output through the 8255A ports. The 8255A interfaces are shown in Fig. 2.3.

We shall discuss the design and schematic of the drive system and interfaces in detail in the next section.

2.2 Drive System and Interfaces

The drive system for each axis consists of a stepper motor, a power driver and a sequence generator. The sequence generator accepts the rate pulses from the 8253 timer and the direction information from one of the ports of the 8255A PIO interface. The stepper motor driver is shown in Fig. 2.4. The sequence generator is shown in detail in Fig. 2.5.

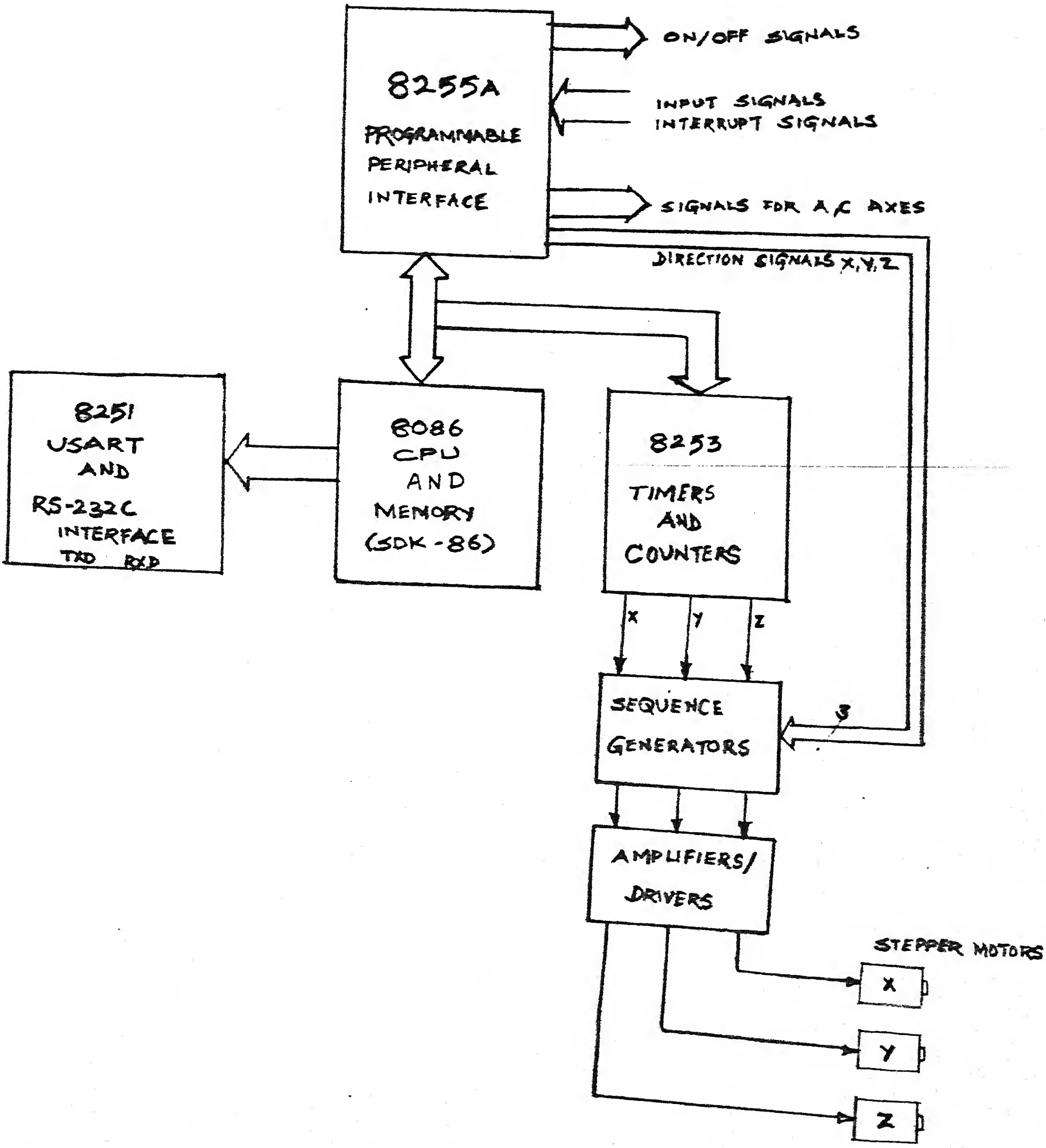


Fig 2.1 BLOCK SCHEMATIC FOR EXPERIMENTAL SETUP

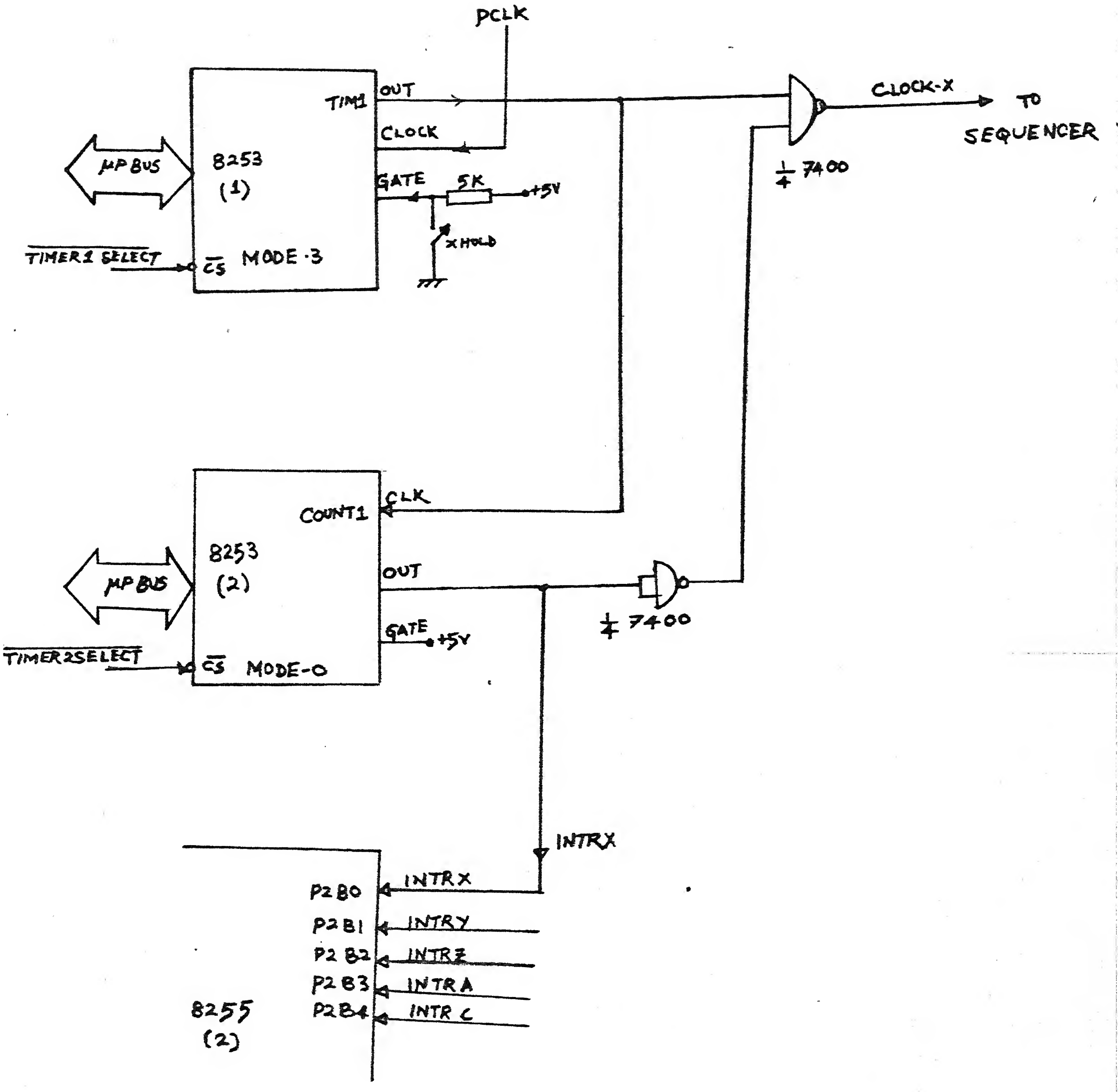

PCLK
TIM1
OUT
CLOCK
CLOCK-X
TO SEQUENCER
μP BUS
8253 (1)
GATE
5K
+5V
XHOLD
TIMER1 SELECT
CS
MODE-3
$\frac{1}{4}$ 7400
COUNT1
CLK
8253 (2)
OUT
μP BUS
GATE
+5V
TIMER2SELECT
CS
MODE-0
$\frac{1}{4}$ 7400
INTRX
P2B0
INTRX
P2B1
INTRY
P2B2
INTRZ
P2B3
INTRA
P2B4
INTRC
8255 (2)

Fig 2.2.

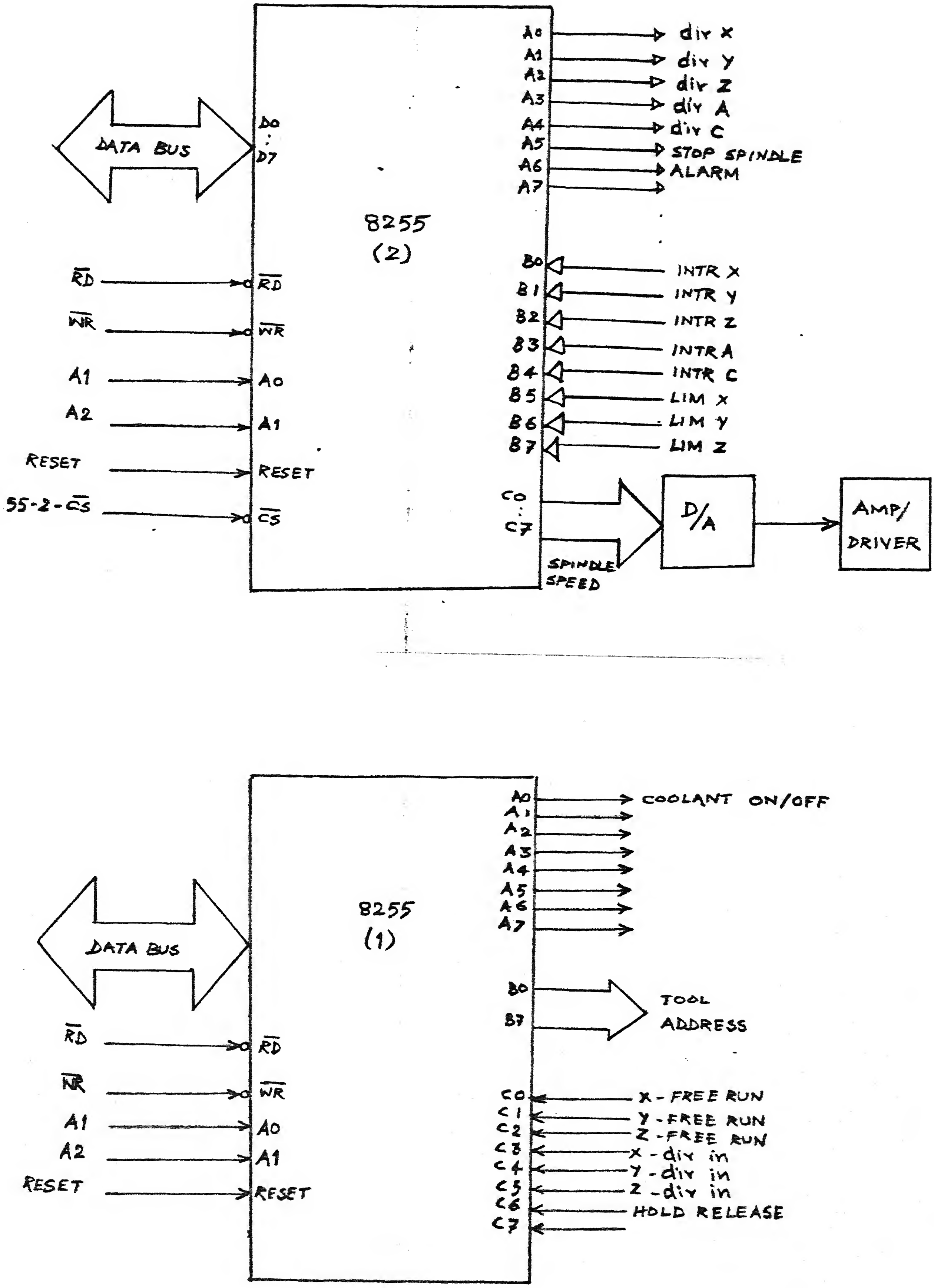


Fig 2.3. Input Output Ports Schematic

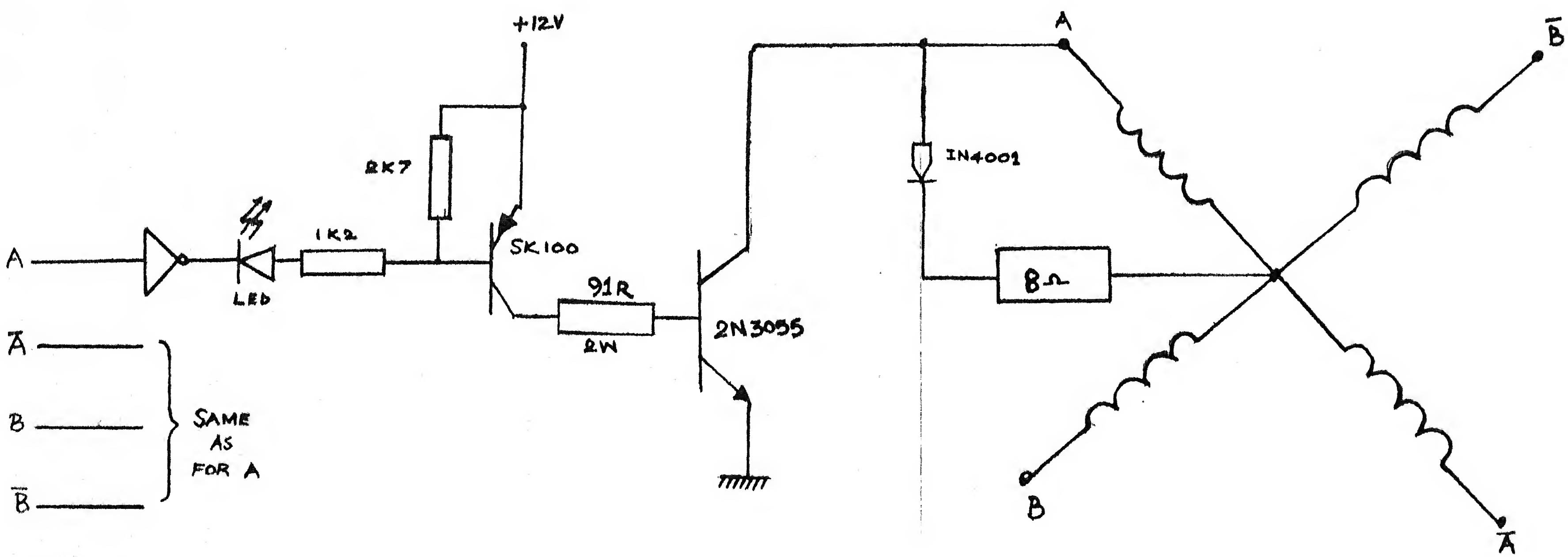


Fig 2.4 STEPPER MOTOR DRIVER

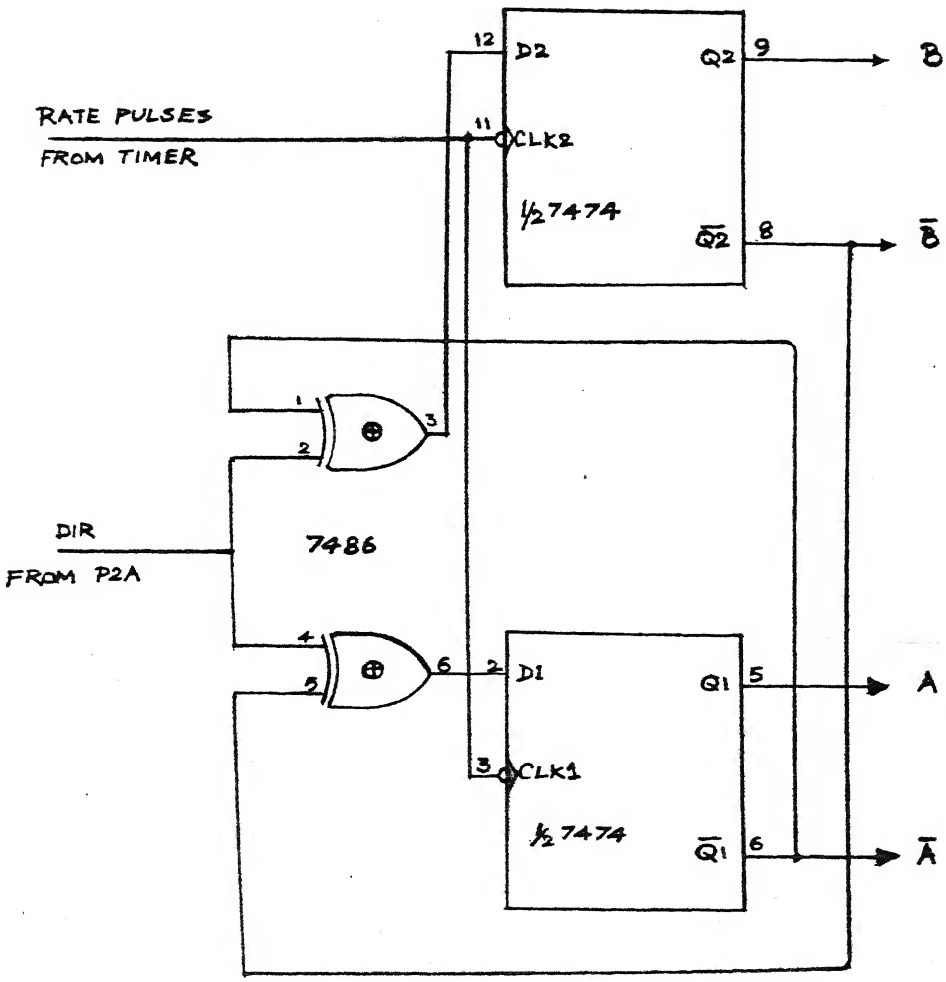


Fig. 2.5 SEQUENCE GENERATOR

The sequence generator consists of two D type flip-flops and two exclusive 'Or' gates as shown in Fig. 2.5. The sequence of excitation for forward (dir = 1) and reverse (dir = 0) is shown in Table 2.1.

Table 2.1

Direction	Dir	Sequence for AB			
Forward	1	00	01	11	10
Reverse	0	00	10	11	01

The sequence generator changes states at the rising edge of the clock pulses coming from the 8253 timer. The 'dir' signal is coming from Port A of 8255A chip No.2. The outputs A and B are inverted and available as $\overline{A}$ and $\overline{B}$ to be amplified by the driver amplifier shown in Fig. 2.4.

The driver amplifier should be capable of handling the maximum load current of the stepper motor.

The power transistor 2N3055 was selected to drive the motor coils which draw a maximum current of about one ampere. To provide the necessary drive current at the base of the power transistor which is around 15m Amp, one SK 100 transistor stage is provided. The SK 100 transistor is driven by the 7406 open

collector inverter. So when the input to the inverter is high, the SK 100 transistor enters saturation. This in turn switches the power transistor ON causing current to flow in the appropriate coil. Due to the high inductance of the coils when switching takes place the current through them cannot vary as fast as the voltage. Hence the transistors may get damaged due to the excessive currents. To overcome this problem, it is mandatory that we provide some alternate path for the currents to flow. Usually this path consists of a diode in series with a dissipating resistor, connected as shown in the circuit diagram in Fig. 2.4.

Since the experimental set-up has a small two axis mechanical table, the motor selected is of 2 kg-cm torque with 200 steps per sec speed and 200 steps per revolution. The stepper motor is used in this system because its positional accuracy is good and no incremental error is involved. The disadvantage is that when a step is applied, the shaft oscillates about the final position before settling down. This problem can be ignored because when pulses are applied in rapid succession, the oscillations do not show up. Also the load tends to dampen the oscillations [6].

The timer/counter interfaces and Input/Output interfaces were discussed in the previous section.

2.3 Plotter Version

Because of mechanical problems resulting from poor alignment of the leadscrews the mechanical table with two axes exhibited considerable amount of friction, it was decided to demonstrate the software with the help of a plotter. The plotter to be used has a RS232-C interface through which our microcomputer can send commands for movement of the pen in X and Y axes. Though the plotter has inbuilt linear interpolator and circular interpolator, to simulate the stepper motor version, use is made only of the incremental commands of the plotter. The command relevant for incremental motion is 'T' or 'D' followed by several ASCII characters as given in the Table 2.2. Use is made only of the +X, -X, +Y, -Y direction movements only. Tool movement in Z direction is restricted only to up/down positions and simulated in this set up as the pen up/down positions.

A second version closely approximating an actual NC system can be implemented using two output ports and two Digital to Analog Converters (DAC). The outputs from these two DAC's can be fed into the X,Y amplifiers of the Servogor plotter which has a feedback loop with the well known potentiometric principle. Use can be made of the DAC 80 (12 bit) chip.

TABLE

Coding of incremental bytes

			size	size	direction					
					↓ 0	← 2	↑ 4	→ 6		
	00	01			02	03	04	05	06	07
00			8	0	SP	0	@	P		
01			9	1	!	1	A	Q		
02			10	2	"	2	B	R		
03			11	3	#	3	C	S		
04			12	4	$	4	D	T		
05			13	5	%	5	E	U		
06			14	6	&	6	F	V		
07			15	7	'	7	G	W		
08			8	0	(	8	H	X		
09			9	1	)	9	I	Y		
10			10	2	*	:	J	Z		
11			11	3	+	;	K	[		
12			12	4	,	<	L	\		
13			13	5	-	=	M	]		
14			14	6	.	>	N	^		
15			15	7	/	?	O	←		
					↙ 1	↖ 3	↗ 5	↘ 7		
					direction					

Table 2.2

Since the program is highly modular the 'MOVE' subroutine only needs to be modified to suit a particular drive type. The original software was modified to suit the plotter version and this shows the flexibility of the system.

2.4 Design Considerations

We have already discussed the considerations which led to the selection of the 16 bit microprocessor intel 8086. The stepper motor drive system was selected because of the non-requirement of feedback since it has no positioning error. The motor has a torque capacity of 2 kg-cm which suffices for our application, i.e., driving the two axis, small milling machine table.

The future modifications and expansion were taken into account while designing the software. The software was made flexible such that it needs only small modifications to suit a new drive system. So the drive system dependant part of the program was to be put into one subroutine so that only that subroutine needs to be modified to suit the change in the drive system configuration. If need arises the whole software can be easily transferred to another type of microprocessor system, say, Zilog 80, because the program was written in PLM-86 language. The selection of PLM language itself was based on the fact that most microprocessor development systems have the PLM compiler.

CHAPTER 3

SOFTWARE DEVELOPMENT

3.1 G-Codes

The concept of the numerical control is that the NC machine is provided with the displacement information by way of numbers [2,3]. But no machine slide can be moved by numerical values alone. There must be some way of telling the NC machine the relevance of each numerical data. This is done by sending an alphabetical letter preceding each numerical information. The first important matter is the symbols for the directions of the axes. Then the information like spindle speed, tool number, feed rate number, miscellaneous functions, preparatory functions etc., are to be identified with a letter called 'address'. The relationships between the letters and machine functions are given in Table 3.1. In addition to the letters there are the decimal numbers and the two signs '+' and '-', to represent the numerical data. Some special symbols and control symbols are required for building up the program. They are listed in Table 3.2. These conventions are from EIA standards [7].

The preparatory function 'G' specifies additional information to the coordinate dimensions and the type of operation

Table 3.1

Symbol	Meaning
A	Rotation about X axis
B	Rotation about Y axis
C	Rotation about Z axis
F	Feed number
G	Preparatory function
I	Interpolation parameter or thread pitch parallel to X axis
J	" " Y axis
K	" " Z axis
M	Miscellaneous function
N	Sequence number
S	Spindle speed function
T	Tool function
X	Movement in the direction of X axis
Y	" " Y axis
Z	" " Z axis

Table 3.2

Special symbols and their meaning

Symbol	Meaning
%	Start of program, also rewind stop
;	Block delimiter
LF	End of sentence
HT	Horizontal tabulator
CR	Carriage return
SP	Space
/	Sentence suppression (Block deleter)
DEL	Character deleter
+	Plus
−	Minus

to be done. The preparatory function thus forms part of the dimensioning. The G code has two digits. The meaning of important G-codes are indicated in the Appendix I.

In addition to the preparatory functions, there are auxiliary functions represented by the address 'M' followed by a two digit code. An auxiliary function is a function of an NC machine other than the control of coordinates of a workpiece or tool. It includes functions such as spindle stop, coolant on, coolant off, clamp, unclamp etc. These are called miscellaneous functions and they are explained in detail in Appendix I.

3.2 Program Description

The entire software is written and debugged in PLM high level language. Because of the modular programming features of PLM, the program consists of relevant modules performing a specific operation, making the program readable.

The flow of the program is as given in the flowchart 1 of the Appendix II. The program listing, alongwith cross reference listing, link map and load map is given in the Appendix III. We shall now discuss the program in detail.

Main Program

On system reset, the program first initializes the various interfaces used in the system. Then the buffers used for

storing the G-code sequence, co-ordinates and other scratch data are initialized.

Then the program control transfers to the subroutine which in turn reads in the G-code sequence in free format, typed in by the programmer through the CRT. After the typing is over, the stored sequence of G-code is displayed for verification. The data entry can be through the tape reader also.

Then the G-code instructions are executed one by one, after receiving a go-ahead signal through CRT. The control branches to any one of the G-code subroutines based on the value of G-function of the current block of code.

In the middle of the program the program may wait for some operator response after displaying some message like requesting change of tool etc. The machine can be stopped in the middle by emergency halt switch. After executing all the blocks, i.e., after encountering an end of file character in the input data, the program asks for instructions from the operator whether another part has to be machined or a new part program to be accepted.

Description of Utility Subroutines

'INITIAL 8251' and 'CHAR$RDY' are procedures used to set the 8251A in asynchronous mode and to find the status of the character buffer of the 8251A, respectively. 'OUT$CHAR'

'GET$CHAR' outputs a character to the monitor and inputs a character from the keyboard respectively. 'OUT$BLANK' outputs a blank or space and 'OUT$CRLF' outputs a carriage return and line feed.

The procedure 'OUT$STRING' is frequency used to display error messages and other interactive instructions in the CRT monitor. This procedure uses a pointer to the message data and keeps on outputting the character pointed to, by the pointer, simultaneously incrementing the pointer, till the data pointed to is 0. This module uses 'OUT$CHAR' and 'OUT$SRLF' procedures.

Function CHK$VALID returns true if the character is an address for a specific function, i.e., N,G,X,Y,Z,A,C,I,J,S,F, T or M, and returns false if it is not a valid function. Procedure 'GET$DATA' stores one block of G-code sequence into the buffer 'BUFFER', which is pointed to by 'INDEX'. The input to this procedure is 'FILE$DATA' with a pointer 'FILE$INDEX'.

Procedure 'INITIAL$DATA' initializes the buffers for the past and new values of the codes and coordinates. Procedure 'STORE$FILE' reads characters from the CRT, with facilities for line editing, and stores them in the 'FILE$DATA' array. Each of the G-code blocks is separated by a semicolon ';', and the entire file is ended by an ESCAPE character.

The procedure GETNEWCODE gets the new values of codes into the respective arrays through the use of procedures 'CODE$3', 'CODE$2' and 'CODE$COORD'. If a character other than the standard codes is encountered, a warning message is given and the processing continues. The most important procedure in the software is the 'ACTION' procedure, which takes appropriate actions according to the G and M functions. This is also the only part of program which needs to be modified to suit another drive system. In the stepper motor version, the action procedure programs the counters and timers of the X,Y,Z,A,C axes and also the direction signals for them. Various auxiliary functions like spindle speed selection, coolant on coolant off, tool changing, clamping etc., are also looked after in this procedure.

Whereas, in the plotter version, instead of the programming of the counters and timers, various characters as required by the plotter (Table 2.2 lists them) are output to the plotter to effect the pen movement in X and Y axes. In the second variation of the plotter version, the X and Y voltages can be given to the plotter instead of the instructions through the plotter's RS 232-C interface. Other functions are programmed as for the stepper motor version.

3.3 Interface to Computer (Software)

The microprocessor based NC machine can communicate with a mainframe computer to receive the G-code sequence directly from the computer. The same software except for some minor modifications can be used to interact with the computer. In that case the CRT terminal will be replaced by the RS232-C port of the computer. The main modifications to be done are as follow :

Firstly the interactive part program entry is to be removed. Also we expect the computer to give error free final version of the G-code sequence. So the checking of the characters as they come through the serial link is not needed. The part of the program which displays the final version of the G-code sequences can be removed. Only the error messages and warning messages, are to be sent to the computer. Functions like tool changing, clamping and unclamping should be automated. Tool number can be output through a port which can be used by the automatic turret to select the specified tool. The workpiece can be clamped automatically by means of electro-magnetic or some other type of clamps. After the machine finishes the machining of the job, the NC machine sends a prompt indicating the computer that it needs instruction for machining one more piece of same part or for receiving new G-code program for a new part. The functions like hold etc.,

can be displayed by means of LED annunciators. The current statement number may also be displayed by means of seven segment LED displays.

3.4 Interface to Computer (Hardware)

The same RS23C-C port can be used for communication with the computer. The connection diagram is as in Fig. 3.1.

An additional port is needed to output the tool address, since the tool has to be changed automatically. For this port B of the 8255A (1) can be used as illustrated in the Fig. 2.2. One more port may be used to indicate the status of the NC machine, viz., Hold, temporary hold, dwell or stop.

3.5 Linear and Circular Interpolation Algorithms

Linear Interpolation : The software DDA algorithm described in [12] is implemented for the linear interpolation. The digital differential analyzer (DDA) method is an interpolation procedure which was developed for use in hardware interpolation. This method requires successive additions in order to create interpolated points and is therefore ideally suited for assembly language simulation. The rate at which the pulses to each axis are sent is controlled by a programmable frequency source, whose output is connected to the interrupt pin, thereby controlling the feed rate to the required level. The algorithm operates as follows, and also as expressed by the flow chart No.3, in Appendix II.

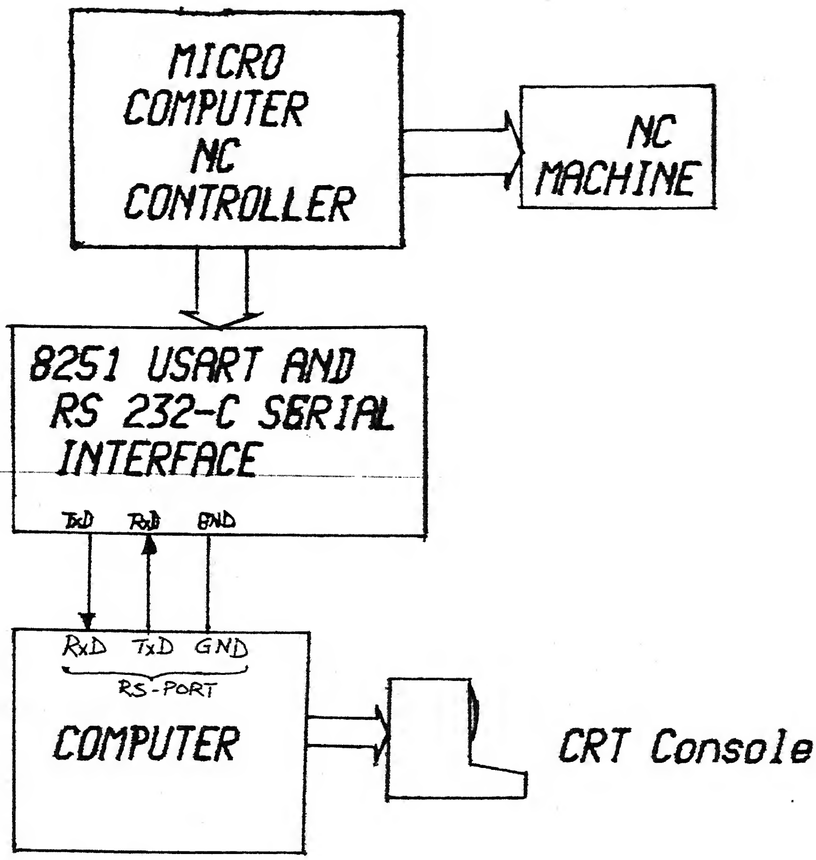


Fig.3.1. Interface to Computer

First, the distances to be moved in X and Y directions (positive or negative) are calculated. The signs of X and Y directions are determined and output. The distances in Basic Length Units (BLU's) are added to give M, which is the total number of pulses to be output in both the directions together. A function f(x,y) is calculated and updated after each pulse is output. This function is initially zero and becomes F+delx or F+dely depending upon the last pulse output is in X direction or in Y direction; delx is equal to -dy where dy is the distance moved in y direction and dely is equal to +dx where dx is the number of pulses to be given to X axis motor (distance moved in BLU's). When the function f changes from negative to positive pulses are given to the Y motor and when the function f changes from positive to negative, pulses are given to the X motor. When all the required pulses are given to the motors, the interpolation ends. This method of linear interpolation is ideally suitable for implementation in assembly language in microcomputers and also for the reference pulse interpolators.

<u>Circular Interpolation :</u> Implementing circular interpolation in a microcomputer controlled system is not easy [8]. It must be done to a guaranteed accuracy regardless of the radius, within an allowed amount of time. Previously the circular interpolation was done in hardware [12], and, with the advent

of microprocessors the interpolation was implemented in software for want of accuracy and flexibility [9,10]. The approach can be in two distinct directions: the reference pulse interpolators [11] or the reference word interpolators [13]. The reference pulse interpolators are simple to implement and are accurate enough. But the reference word interpolators are complex if accuracy is needed. Since one segment rather than the immediate neighbour point of the curve is calculated at each iteration, the curve is generated with fewer iterations, though the iteration itself may take more processor time. Since our need is limited to a few thousand BLU's per second, (which is due to the nonavailability of faster step motors with higher torque), the first method, i.e., reference pulse interpolators is implemented in our system. A detailed study of various interpolators of the first method had been done in [11]. A simple algorithm for the second type of interpolation is described in [8]. Of the four types of reference pulse interpolators that Koren, Y [11] describes, the improved direct search method was found to be accurate and also appropriate for our system and hence it has been implemented in our system. The error involved is 1/2 BLU maximum. The algorithm first evaluates the direction of each axis movement and the distance to be moved and the centre of the circular arc. A function D, similar to that used in linear interpolation is evaluated for three possible increments: in X direction, in Y direction and,

in X and Y direction. This function is proportional to the errors caused because of the corresponding movements. Then decision is taken as to which path is to be taken, based on the three error functions d_1, d_2 and d_3. Then the pulses are sent to the appropriate axes. This completes one iteration. The iterations end when the computed point passes [10] the final point. The algorithm is illustrated by means of the flow chart 4 of the appendix II.

CHAPTER 4

TEST EXAMPLES

4.1 Example No.1 (Point-to-point positioning)

This example illustrates how the NC system can be used as a point-to-point positioning machine, i.e., like a drilling machine. The following G-code sequence is typed in at the CRT console.

```
NO1 GOO ;
NO2 G91xO Y200 ;
NO3 X300 Y400 ;
NO4 X300 YO ;
NO5 X500 Y-400 ;
NO6 XO Y-200 ;
NO7 X-1100 YO MO2;
```

The first statement informs the machine that point-to-point programming mode be set, through the G-code GØØ. The second statement sets the machine in the incremental dimensioning mode (G91). The series of co-ordinates command the tool to move to the required points. The seventh statement implies that end of program is arrived (through the miscellaneous code MO2).

The plotter output corresponding to the above set of commands is given in Fig. 4.1. One can see that the movement is such that it arrives at the end point in shortest possible time.

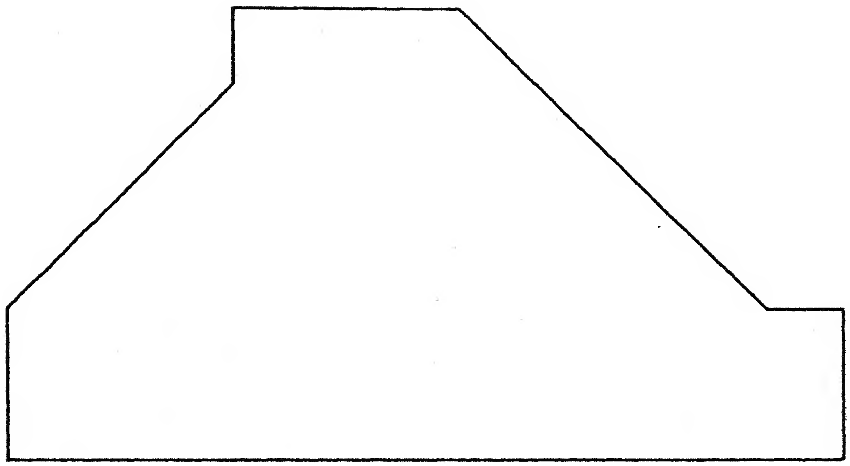


Fig. 4.1. Point-to-point positioning

4.2 Example No.2 (Linear Interpolation)

The second example illustrates the application of this NC system as a contouring milling machine with straight line cuts. The following G-code instructions are typed into the CRT console.

```
NØ1 GO1 ;
NØ2 G91 X400 Y500 Z-50 ;
NO3 XO Y200 Z50 ;
NO4 X400 Y200 ;
NO5 X400 YO;
NO6 X60 Y-400
NO7 X-200 YO ;
NO8 X100 Y150;
NO9 X-300 YO ;
N10 X100 Y-150 ;
N11 X-500 YO ;
N12 MO2 ;
```

The first statement informs the NC system that the movement of the tool should be in linear interpolation mode. The second line has G91 code which implies that the incremental dimensioning was chosen. Then the successive endpoints . coordinates are given in the next lines. The last line has a miscellaneous function MO2, which informs the machine that the end of program is arrived.

The plotter output corresponding to the above G-code sequence is given in Fig. 4.2. A step size in each direction of 0.1 mm is used in the system.

4.3 Example No.3 : (Circular Interpolation)

The final example illustrates the system as a circular interpolating contouring NC machine. The following G-code sequence is typed into the CRT console.

```
N01 G00 X400 Y500 Z50 ;
N02 G01 X0   Y200 Z50 ;
N03 X200 Y0 ;
N04 G02 X200 Y200 I200 J0 ;
N05 G02 X200 Y200 I0   J200 ;
N06 G01 X200 Y0 ;
N07 X0  Y-200 ;
N08 X800 Y0 ;
N09 M02 ;
```

The first statement positions the tool (pen) at bottom left corner of the workpiece. Statements 2 and 3 move the tool in linear interpolation mode. The fourth statement commands the machine tool to take a clockwise circular path. This moves the tool through one quadrant of the circle. The next statement moves the tool through another quadrant. Statements 6 through 8 moves the tool in linear interpolation mode. The machine stops when it encounters the program end (M02) auxiliary function.

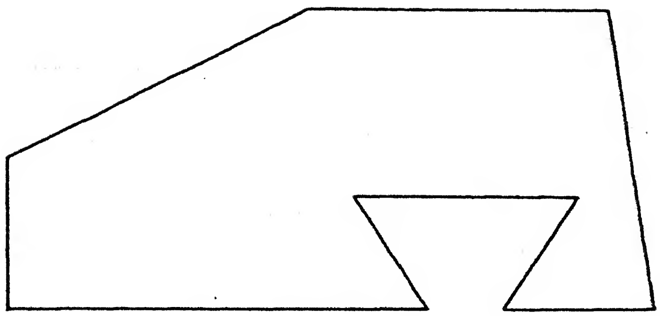


Fig. 4.2 Linear Interpolation

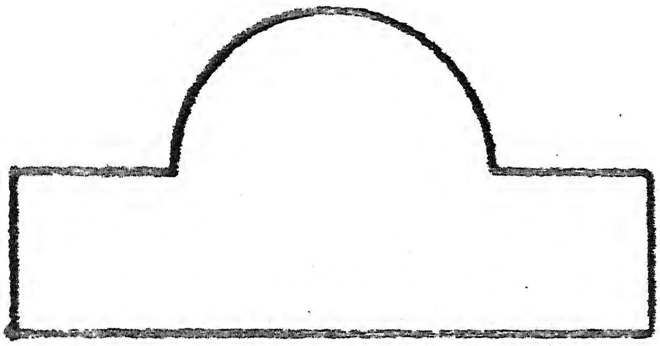


Fig. 4.3 Circular Interpolation

The plotter output corresponding to the above example is given in Fig. 4.3. It is to be noted that at a time the tool can be commanded to move through only one quadrant of a circle. The direct search method and other reference pulse circular interpolation methods have this limitation in common.

CHAPTER 5

CONCLUSIONS AND SUGGESTIONS

6

5.1 Technical Summary

The hardware and the software were tested successfully for the NC machine. The point-to-point positioning, linear interpolation and the circular interpolation algorithms are found to be suitable to implement on a reference pulse CNC machine. Because of nonavailability of a stepper motor controlled NC milling table, it was decided to simulate the machine by a plotter. Use was made only of the plotter's +X, -X, +Y, -Y incremental mode commands and this approximated a stepper motor controlled NC milling table.

The software is written in appropriate modules to suit future modifications. The whole program controlling the NC machine has been written in PLM-86 rather than in assembly language of intel 8086, to facilitate partability. So, if need arises the system can be re-modified to suit another microprocessor. Since it was not intended to drive the plotter in real-time (since the communication with the plotter is through a slow serial link), the reference pulse interpolation [11] or the reference word interpolation [13] technique was not really implemented, in the sense, that the microprocesssor is not interrupted by the reference pulses at a frequency proportional to

the feed rate. This can be easily done with a programmable frequency source whose output is tied with the interrupt pin (INTR) and suitable changes in software. The interpolation routines are clustered into the subroutine MOVE, so that this subroutine can be used as the interrupt service routine. Essentially the movement in X,Y or Z should be made in synchronism with the reference pulses. A detailed study of the techniques involved is found in [11].

5.2 Suggestions for Further Work

The main theme of this project has been to design a full-fledged CNC work station. The present work is the first step towards the work station concept. The gap between the small CNC work station and large CNC centers is becoming narrower day by day [9], with miniaturisation and cost reduction in electronics. More and more facilities are included in the small work stations controlling a single NC machine. Based on the results of the present work the following suggestions are made for further work, to bring about the CNC machining center concept to practice :

(1) An interrupt structure can be designed to implement the reference pulse interpolation, on a true multiaxis milling table.

(2) Presently the circular interpolation (GO2, GO3) is implemented in one plane only. The third simultaneously controlled axis can be also moved during the circular motion of the other two axes, to implement helical interpolation.

(3) An algorithm can be devised to control the three axes simultaneously using DDA method. Mayorov F.V. [12] gives insight into the DDA concepts.

(4) A feedback loop can be incorporated with the instalment of resolvers and up/down counters to drive a dc motor controlled machine, with the help of D to A converters and appropriate drive amplifiers.

(5) A floppy disk controller can be added to store the part programs. Also, more facilities can be included in the text editor.

(6) With more memory added in the microcomputer system, a high level part programming language can be implemented in the system.

The above suggestions are only a few of the many possible ideas for future work.

APPENDIX I

G-CODE CONVENTIONS

Address Functions	
Character	Function
A	Angular dimension around X-axis
B	Angular dimension around Y-axis
C	Angular dimension around Z-axis
D	Specifies tool offset and cutter compensation offset number
F	Specifies feed rate
G	Specifies operation mode (linear, circular etc.,)
H	Auxiliary function (ON/OFF controls)
I	Interpolation parameter along X-axis
J	'' '' Y-axis
K	'' '' Z-axis
L	Subprogram number
M	Auxiliary function (ON/OFF controls)
N	Sequence number
S	Specifies revolution speed of main spindle
T	Specifies a tool number
X	Basic co-ordinate axis (X-axis)
Y	'' (Y-axis)
Z	'' (Z-axis)

Preparatory Functions (G-functions)	
Code	Function
G00	Point-to-point, positioning at max speed
G01	Linear interpolation at specified feed rate
G02	Circular interpolation (clockwise)
G03	Circular interpolation (counter clockwise)
G04	Dwell (for a specified duration-X seconds)
G09	In-position check
G28	Automatic return to zero
G29	Automatic return to zero through a specified point
G40	Cutter compensation cancel
G41	Cutter compensation to the left-side of tool path
G42	Cutter compensation to the right-side of tool path
G43	Tool axis offset (+ offset)
G44	Tool axis offset (- offset)
G53	Programming in basic machine co-ordinate system
G54	Selection of work co-ordinate system 1
G55	'' 2
G60	In-position check mode
G63	Trapping mode
G64	Cutting mode
G90	Absolute input mode
G91	Incremental input mode
G92	Setting up new work co-ordinate system

G-code Format used in this system

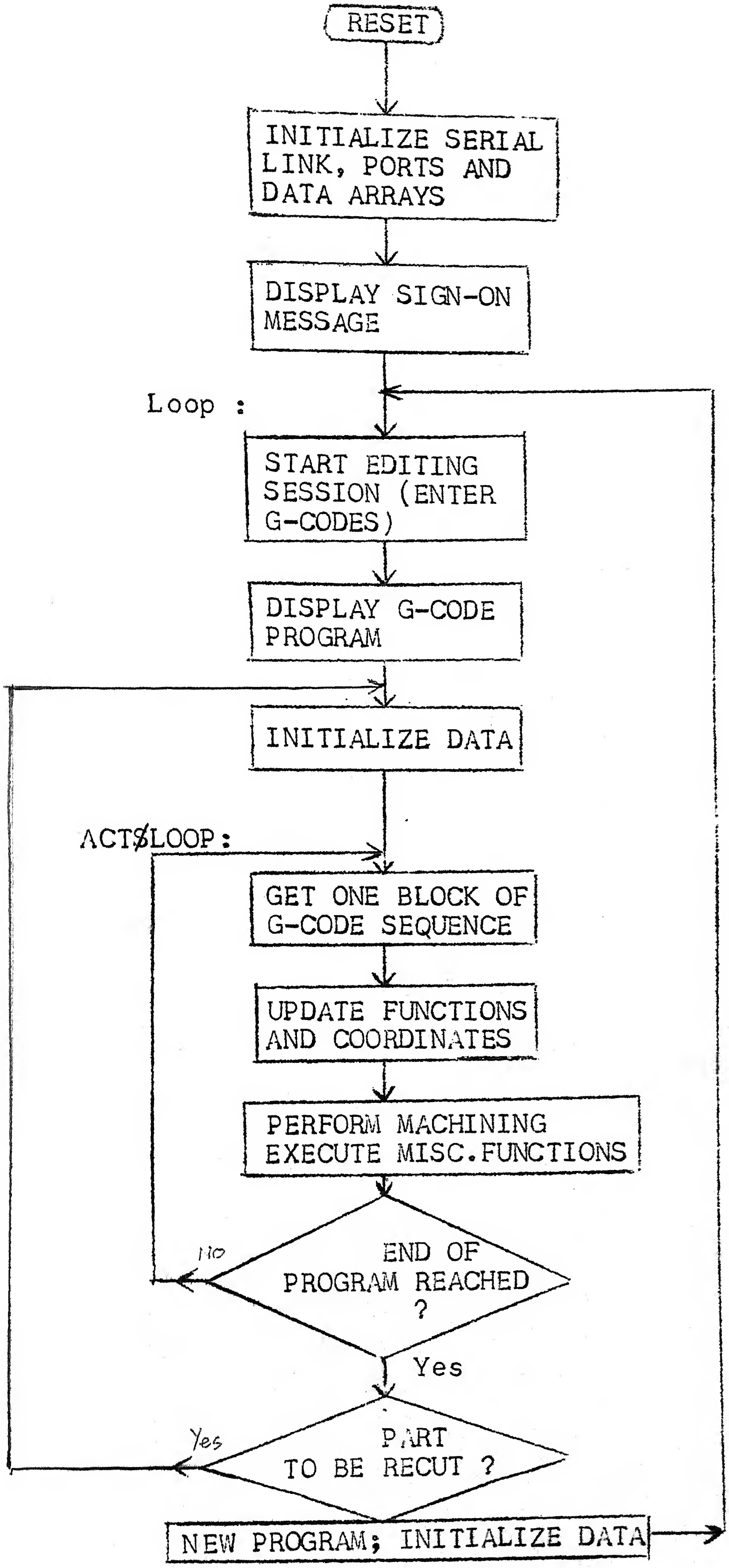
The user should type in the G-code blocks each separated by a semicolon (';'). Any blanks, linefeed, carriagereturns and illegal characters are ignored and hence not stored. The user ends editing by pressing escape <ESC> key.

Within a block, user can type in the addresses and corresponding data in any order. The only restriction is that, an address function should be followed by digits only, whose length should not exceed 5 for co-ordinates, 3 for feedrate, spindle-speed functions and 2 for other valid functions.

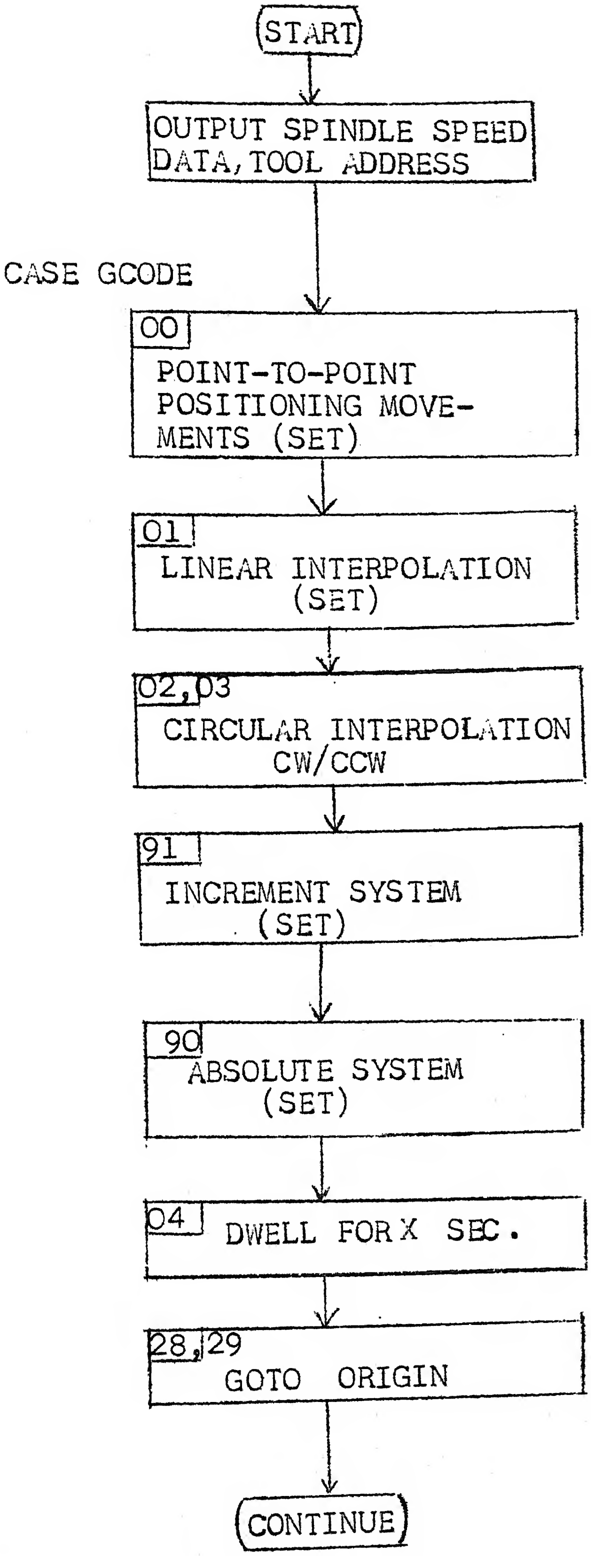
This free format is to facilitate easy entry of the co-ordinates, functions and codes.

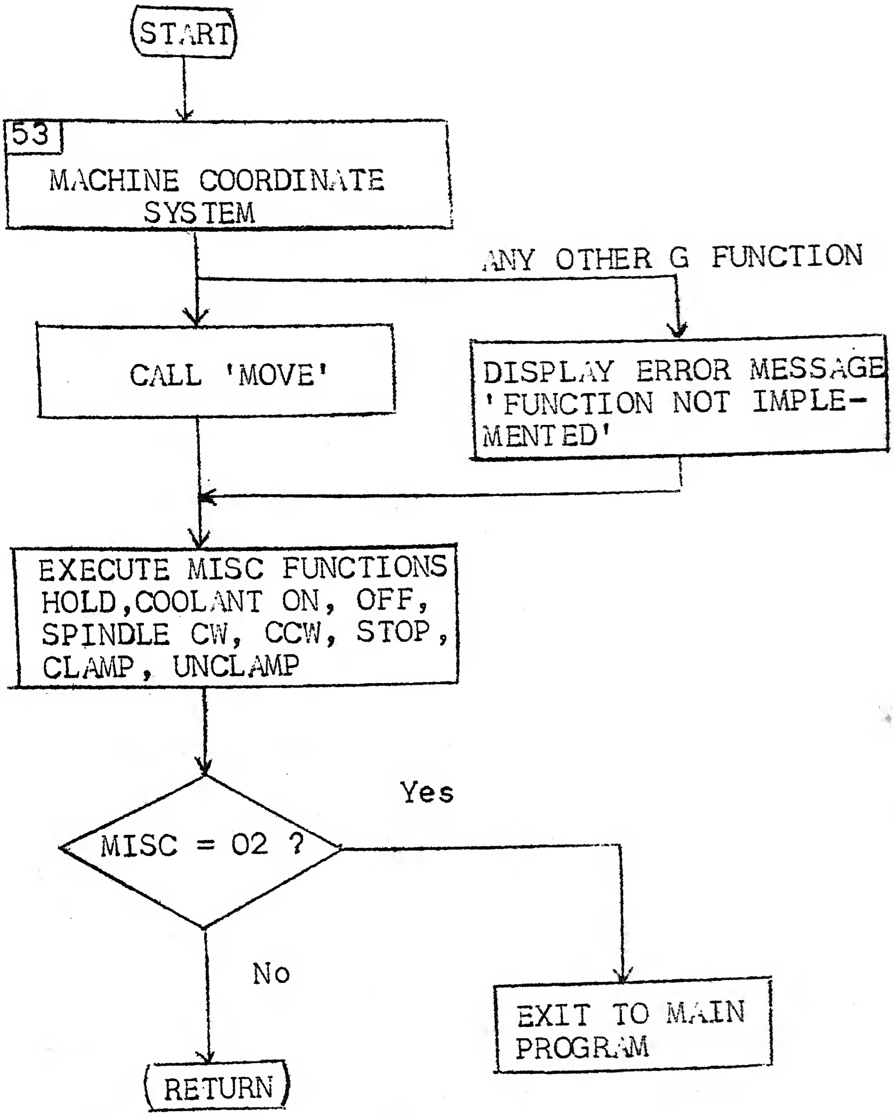
MISCELLANEOUS FUNCTIONS

Code	Function
M00	Program stop (Hold)
M01	Optional (planned) stop
M02	End of program
M03	Spindle clockwise
M04	Spindle Counter Clockwise
M05	Spindle OFF
M06	Tool change
M07	Coolant No.2 ON
M08	Coolant No.1 ON
M09	Coolant OFF
M10	Clamp
M11	Unclamp
M13	Spindle CW and Coolant ON
M14	Spindle CCW and Coolant ON
M30	End of Tape
M31-99	Optional and unassigned

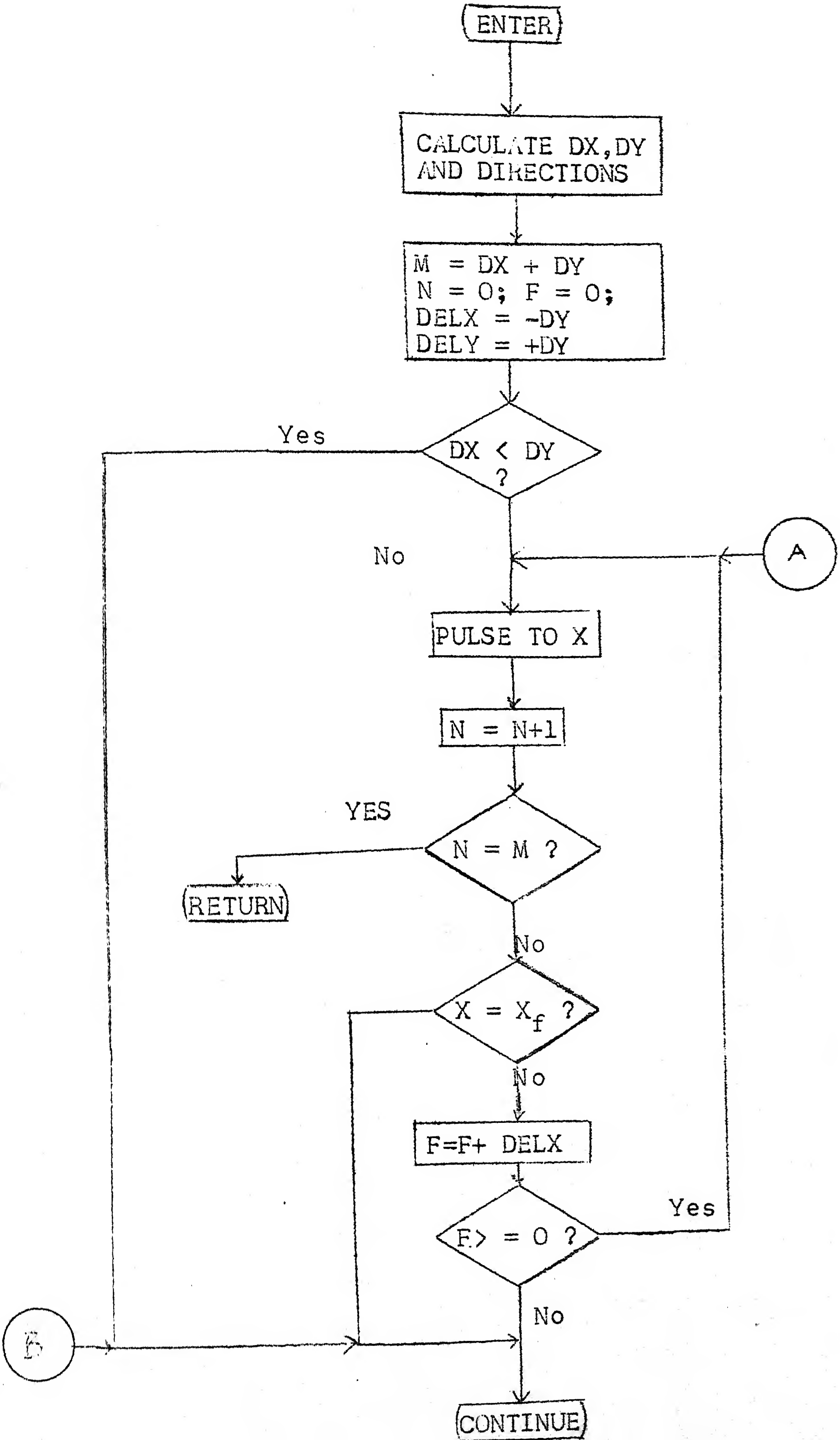


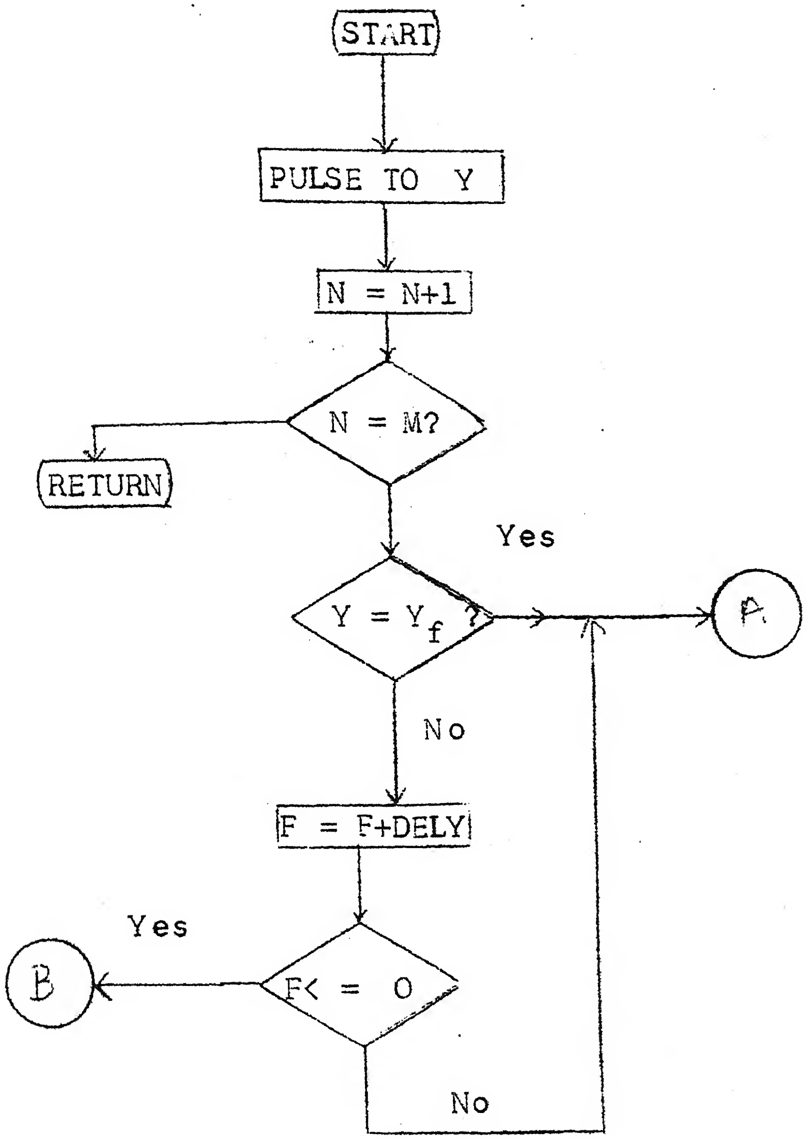
Flow Chart 1 MAIN PROGRAM


START
OUTPUT SPINDLE SPEED DATA, TOOL ADDRESS
CASE GCODE
00
POINT-TO-POINT POSITIONING MOVEMENTS (SET)
01
LINEAR INTERPOLATION (SET)
02,03
CIRCULAR INTERPOLATION CW/CCW
91
INCREMENT SYSTEM (SET)
90
ABSOLUTE SYSTEM (SET)
04
DWELL FOR X SEC.
28,29
GOTO ORIGIN
CONTINUE

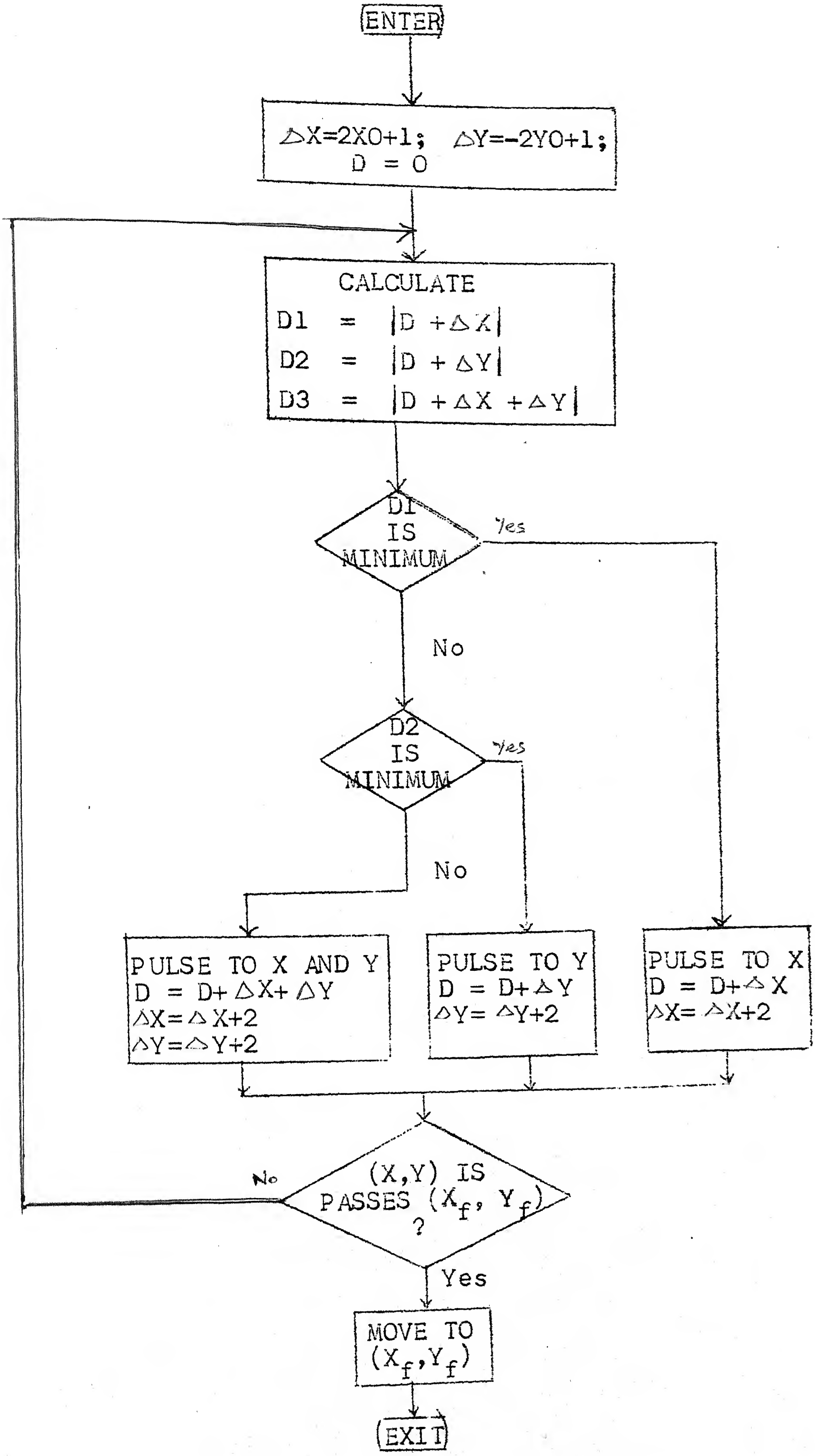


Flow Chart 2 PROCEDURE 'ACTION'


ENTER
CALCULATE DX,DY AND DIRECTIONS
M = DX + DY
N = 0; F = 0;
DELX = -DY
DELY = +DY
DX < DY ?
Yes
No
A
PULSE TO X
N = N+1
YES
N = M ?
RETURN
No
X = X_f ?
No
F=F+ DELX
Yes
F> = 0 ?
No
B
CONTINUE



Flow Chart 3 LINEAR INTERPOLATION ALGORITHM


ENTER
ΔX=2X0+1; ΔY=-2Y0+1;
D = 0
CALCULATE
D1 = |D + ΔX|
D2 = |D + ΔY|
D3 = |D + ΔX + ΔY|
D1 IS MINIMUM
Yes
No
D2 IS MINIMUM
Yes
No
PULSE TO X AND Y
D = D+ ΔX+ ΔY
ΔX= ΔX+2
ΔY= ΔY+2
PULSE TO Y
D = D+ ΔY
ΔY= ΔY+2
PULSE TO X
D = D+ ΔX
ΔX= ΔX+2
No
(X,Y) IS PASSES (Xf, Yf) ?
Yes
MOVE TO (Xf,Yf)
EXIT

```
III PL/M-86 V2.0 COMPILATION OF MODULE NCM
MODULE PLACED IN THESIS.OBJ
  INVOKED BY:   :F2:PLM86.86 THESIS.PLM



        $TITLE ('NC M/C CONTROLLER')
        $LIST
        $XREF
        $PAGELENGTH(50)
        $PAGEWIDTH(70)


         NCM:DO;
         /*GLOBAL DATA DECLARATIONS*/
1        DECLARE INT$VECTOR(5) POINTER;
1        DECLARE CR LITERALLY '0DH',
                 LF LITERALLY '0AH';
1        DECLARE
              SIGN$ON(*) BYTE DATA (CR,LF,'NC Machine Controller
              - -Plotter version',0),
              ABORT(*) BYTE DATA ('^Z',CR,LF,'Run aborted',0),
              ERR1(*) BYTE DATA('Function not implemented',0),
              ERR2(*) BYTE DATA('Illegal character found',0);
1        DECLARE TRUE LITERALLY 'OFFH',
                 FALSE LITERALLY '000H',
          MESS1(*) BYTE DATA ('ENTER THE G-CODE SEQUENCE WITH EA
              -CH BLOCK SEPARATED',0),
          MESS2(*) BYTE DATA ('BY A SEMICOLON ; AND TO END EDITI
              -NG,PRESS <ESC> KEY',0),
          MESS3(*) BYTE DATA(CR,LF,'ILLEGAL CHARACTER ENTERED;FO
              -R EXIT PRESS <ESC>',CR,
                                          LF,0),
          MESS4(*) BYTE DATA (CR,LF,'TYPE N TO START A NEW PROGR
              -AM,C TO MACHINE AGAIN',0);

1       DECLARE ERR5(*) BYTE DATA ('Unexpected character found',
              -0),
                WARN(*) BYTE DATA ('WARNING: Character ignored',
              -0);

1        DECLARE VALID$FN(*) BYTE DATA ('NGXYZACSFMTIJ');/*VALID
              - FUNCTIONS' ADDRESS*/
1        DECLARE DIGIT(*) BYTE DATA('0123456789');
1        DECLARE (I,J,K,L,M,N,II) BYTE,
                                  (FL,CHAR) BYTE;
1        DECLARE FILE$INDEX WORD;
```

```
      1         DECLARE FILE$DATA (1000) BYTE;

      1         DECLARE   /*Port numbers declarations*/
                        STAT$8251 LITERALLY '0FFF2H',/*8251 STATUS
                - PORT*/
                        DATA$8251 LITERALLY '0FFF0H',/*8251 DATA P
                -ORT*/
                        RESET$8251 LITERALLY '065H',/*8251 RESET C
                -OMMAND*/
                        MODE$8251 LITERALLY '0CFH',/*8251 ASYNC MO
                -DE SET COMMAND*/
                        CMND$8251 LITERALLY '025H',/*ENABLE TX,RX*
                -/
                        DS$RDY LITERALLY '080H',/*DSR READY MASK*/
                        RX$RDY LITERALLY '02H',/*RX READY MASK*/
                        TX$RDY LITERALLY '01H';/*TX READY MASK*/

      1         DECLARE   /*ASCII BYTE VALUES -CONST DECLARATIONS*/
                        BLANK LITERALLY '20H',
                        BKSPCE LITERALLY '08H',
                        EOB LITERALLY '3BH',
                        SCRNBLNK LITERALLY '0CH',
                        DELETE LITERALLY '010H';
      1         DECLARE    /*BUFFERS DECLARATIONS*/
                        CODE$PAST(13) INTEGER,
                        CODE$NOW(13) INTEGER,
                        BUFFER(128) BYTE,
                    T(5) INTEGER INITIAL (0,0,0,0,0);
      1        DECLARE INDEX BYTE;

      1         DECLARE        TEMP INTEGER;

      1         DECLARE    /*BOOTING OPCODE STORE DECLARATIONS*/
                        START$ADDR LITERALLY '0000H',
                        BOOT1(*) BYTE AT (0FFFF0H) DATA (0EAH),/*L
                -ONG JUMP OPCODE*/
                        BOOT2(*) WORD AT (0FFFF1H) DATA (START$ADD
                -R),/*OFFSET ADDRESS*/
                        BOOT3(*) WORD AT (0FFFF3H) DATA (0FF00H);/
                -*SEGMENT ADDRESS*/

      1        DECLARE P2ABUF BYTE,
                       P2BBUF BYTE,
                       P2CBUF BYTE;

      1        DECLARE P2A LITERALLY '0FFF8H',
                       P2B LITERALLY '0FFFAH',
```

```
                 P2C LITERALLY 'OFFFCH',
                 P2S LITERALLY 'OFFFEH';

1         DECLARE INCREMENTAL BYTE,
                  LINEAR BYTE,
                  CIRCULAR BYTE,
                  CWISE BYTE;


          /* FOR INITPORT PROCEDURE - GLOBAL VARIABLES */

1         DECLARE COOLNT LITERALLY '01H',
              P1A LITERALLY 'OFFF9H',
              P1B LITERALLY 'OFFFBH',
              P1C LITERALLY 'OFFFDH',
              P1S LITERALLY 'OFFFFH';



1         DECLARE P1ABUF BYTE,
              P1BBUF BYTE,
              P1CBUF BYTE;

                 /* INITIALIZATION PROCEDURES */


1         INITPORT: PROCEDURE;
2         OUTPUT(P1S)=82H; /*P1A=OUTPUT, P1B=INPUT, P1C=OUTPUT */
2         P1ABUF=OFEH;
2         P1BBUF=00;
2         P1CBUF=00;
2         OUTPUT(P1A)=P1ABUF;
2         OUTPUT(P1C)=P1CBUF;
2         END;

1         INITIALPORTS: PROCEDURE;
2         OUTPUT(P2S)=82H;
2         P2ABUF=9FH;
2         OUTPUT(P2A)=P2ABUF;
2         P2CBUF=00H;
2         OUTPUT(P2C)=P2CBUF;
2         END;

                       /*LOW LEVAL PROCEDURES */

1           INITIAL$8251: PROCEDURE;
```

```
 39   2        OUTPUT(STAT$8251)=65;CALL TIME(50);
 41   2        OUTPUT(STAT$8251)=25;CALL TIME(50);
 43   2        OUTPUT(STAT$8251)=RESET$8251;
 44   2        CALL TIME(50);
 45   2        OUTPUT(STAT$8251)=MODE$8251;
 46   2        CALL TIME(50);
 47   2        OUTPUT(STAT$8251)=CMND$8251;
 48   2        CALL TIME(50);
 49   2        END;

 50   1        CHAR$RDY:PROCEDURE BYTE;
 51   2        IF (INPUT(STAT$8251) AND RX$RDY)=0 THEN RETURN FALSE;
 53   2        ELSE RETURN TRUE;
 54   2        END;

 55   1              OUT$CHAR:PROCEDURE(CH);
 56   2              DECLARE CH BYTE;
 57   2              IF (CHAR$RDY) THEN CALL CHK$CTL$CHAR;
 59   2              DO WHILE(INPUT(STAT$8251) AND TX$RDY)=0;
 60   3              END;
 61   2              OUTPUT(DATA$8251)=CH;
 62   2              END;

 63   1              GET$CHAR:PROCEDURE;
 64   2              DO WHILE(INPUT(STAT$8251) AND RX$RDY)=0;
 65   3              END;
 66   2              CHAR=INPUT(DATA$8251)AND (07FH);
 67   2              IF CHAR>=BLANK THEN CALL OUT$CHAR(CHAR);
 69   2              IF CHAR=CR THEN CALL OUT$CRLF;
 71   2              END;


 72   1              OUT$CRLF:PROCEDURE;
 73   2              CALL OUT$CHAR(CR);
 74   2              CALL OUT$CHAR(LF);
 75   2              END;

 76   1              OUT$STRING: PROCEDURE(PTR);
 77   2              DECLARE PTR POINTER;
 78   2              DECLARE STR BASED PTR(1) BYTE ;
 79   2              DECLARE IND BYTE ;
 80   2              IND=0;
 81   2              DO WHILE STR(IND)<>0;
 82   3              CALL OUT$CHAR(STR(IND));
 83   3              IND=IND+1;
 84   3              END;
 85   2              CALL OUT$CRLF;
```

```
 86   2          END;

 87   1          CHK$VALID:PROCEDURE BYTE;
                         /*INPUT TO THIS PROCEDURE IS THE GLOBAL VA
                 -RIABLE CHAR*/
 88   2          DO I=0 TO LAST(VALID$FN);
 89   3          IF CHAR=VALID$FN(I)THEN RETURN TRUE;
             /*Returns the index I,the subscript of the function in
                 -the array*/
 91   3          END;
 92   2          RETURN FALSE;
 93   2          END;

 94   1          GET$DATA:PROCEDURE;
                         /*STORES ONE BLOCK OF G-CODE SEQUENCE INTO
                 - A BUFFER*/
 95   2          DECLARE IND BYTE;
 96   2          IND=0;
 97   2          IF (FILE$DATA(FILE$INDEX)=0) OR (FL=1) THEN GOTO E
                 -XIT;
 99   2          DO WHILE FILEDATA(FILE$INDEX)<> (';');
100   3          BUFFER(IND)=FILE$DATA(FILE$INDEX);
101   3          IF FILE$DATA(FILE$INDEX)=0H THEN
102   4                         DO;BUFFER(IND)=0;FL=1;GOTO EXT$LOOP
                 -;END;
107   3          IND=IND+1;
108   3          FILE$INDEX=FILE$INDEX+1;
109   3          END;
110   2          BUFFER(IND)=0;
111   2          FILE$INDEX=FILE$INDEX+1;
112   2          END;

113   1          INITIAL$DATA:PROCEDURE;
                         /*Initialise CODE$PAST,CODE$NOW,FILEINDEX
                 -*/
114   2          FILE$INDEX=0;
             /* Sets the machine in incremental,linear interpolation
                 - mode */
115   2          INCREMENTAL=TRUE;
116   2          LINEAR=TRUE;
117   2          CIRCULAR=FALSE;
118   2          CWISE=TRUE;
119   2          FL=0;
120   2          DO I=0 TO 12;
121   3          CODE$PAST(I)=0;
122   3          CODE$NOW(I)=0;
123   3          END;
```

```
124   2        CODE$NOW(1)=90;/*G FUNCTION ABSOLUTE DIMENSION */
125   2        CODENOW(9)=99;/* MISC */
126   2        END;

127   1        STORE$FILE:PROCEDURE;
128   2        DECLARE LST$BLK WORD,
                        BLK BYTE;
129   2        FILE$INDEX=0;LST$BLK=0;
131   2        BLK=0;
132   2        CALL OUT$STRING(@MESS1);
133   2        CALL OUT$STRING(@MESS2);
134   2        LOOP:CALL GET$CHAR;
135   2        IF ((CHAR>=041H) AND (CHAR<=05AH)) THEN
136   2         DO;
137   3          FILE$DATA(FILE$INDEX)=CHAR;
138   3          FILE$INDEX=FILE$INDEX+1;
139   3         END;
140   2        ELSE IF ((CHAR>=061H) AND (CHAR<=07AH)) THEN
141   2         DO;
142   3          FILE$DATA(FILE$INDEX)=CHAR-020H;
143   3          FILE$INDEX=FILE$INDEX+1;
144   3         END;
145   2        ELSE IF ((CHAR>=030H)AND(CHAR<=39H)) OR ((CHAR='+'
               -) OR (CHAR='-')) THEN
146   2         DO;
147   3          FILE$DATA(FILE$INDEX)=CHAR;
148   3          FILE$INDEX=FILE$INDEX+1;
149   3         END;
150   2        ELSE IF ((CHAR=BKSPCE)OR(CHAR=DELETE)) THEN
151   2         DO;
152   3          FILE$INDEX=FILE$INDEX-1;
153   3          IF FILE$INDEX=-1 THEN FILE$INDEX=0;
155   3         END;
156   2        ELSE IF (CHAR=01BH) THEN
157   3         DO;FILE$DATA(FILE$INDEX)=0;RETURN;END;  /*<ESC> EN
               -COUNTERED */
161   2        ELSE IF (CHAR=BLANK)OR (CHAR=LF)OR(CHAR=CR) THEN
162   2         GOTO LOOP;
163   2        ELSE IF(CHAR=';')THEN
164   2         DO;
165   3          FILE$DATA(FILE$INDEX)=CHAR;
166   3          LST$BLK=FILE$INDEX;
167   3          BLK=BLK+1;
168   3          FILE$INDEX=FILE$INDEX+1;
169   3         END;
170   2        ELSE IF (CHAR=015H) THEN     /* IF CHAR=CTL U */
171   2         DO;
```

```
172   3              FILE$INDEX=LST$BLK;
173   3              CALL OUT$CRLF;
174   3             END;
175   2            ELSE
                    DO;
176   3             CALL OUTSTRING(@MESS3);
177   3             END;

178   2      IF FILE$INDEX > 998 THEN DO;
180   3          DECLARE LIMITERR(*) BYTE DATA('File limit exceeded;
                   -EOF inserted. ',0);
181   3          CALL OUT$STRING(@LIMITERR);
182   3          FILE$DATA(FILE$INDEX)=0;
183   3          RETURN;
184   3          END;
185   2            GOTO LOOP;
186   2            END;

187   1            CHK$CTL$CHAR:PROCEDURE;
188   2            CHAR=INPUT(DATA$8251) AND 07FH;
189   2            IF CHAR=13H THEN /* CTL S? */
190   3             DO WHILE (CHAR<>11H); /* CTL Q? */;
192   3              IF CHAR$RDY THEN
193   3                DO;
194   4                CHAR=INPUT(DATA$8251) AND 07FH;
195   4                IF CHAR=1AH THEN GOTO CTLZ;/* CTL Z */
197   4                END;
198   3             END;/* DO WHILE */
199   2              ELSE IF CHAR=1AH THEN GOTO CTLZ;
201   2            END;/* CHK$CTL$CHAR */

202   1      ACTION:PROCEDURE;
203   2      DECLARE CODE INTEGER,
                  MISC INTEGER,
                  SPEED INTEGER,
                  TOOL INTEGER,
                  TOOLMESS(*) BYTE DATA ('Automatic tool change being
                    -done',0);

204   2      DECLARE MAXSPEED INTEGER DATA (125);

205   2      OUTINTEGER:PROCEDURE(NUMBER);
206   3      DECLARE NUMBER INTEGER;
207   3      P1CBUF=LOW(UNSIGN(NUMBER));
208   3      OUTPUT(P1C)=P1CBUF;
209   3          END;
```

```
210    2     HOLD: PROCEDURE;
211    3     REP: I=INPUT(P1B);
212    3     I=I AND (40H) ;
213    3     IF I=0 THEN GOTO REP;
215    3         END;

216    2     NOTIMPLEM: PROCEDURE;
217    3     CALL OUTSTRING(@ERR1);
218    3         END;

219    2     DELAY: PROCEDURE;
220    3     DECLARE N WORD;
221    3     DO N=1 TO CODENOW(2);
222    4         CALL TIME(125);
223    4     END;
224    3     CODENOW(2)=CODEPAST(2);

225    3     END;

226    2     MOVE: PROCEDURE;
             /* LOCAL VARIABLES DECLARATIONS */
227    3     DECLARE X2 LITERALLY 'CODENOW(2)',
                 Y2 LITERALLY 'CODENOW(3)',
                 Z2 LITERALLY 'CODENOW(4)',
                 X1 LITERALLY 'CODEPAST(2)',
                 Y1 LITERALLY 'CODEPAST(3)',
                 Z1 LITERALLY 'CODEPAST(4)';

228    3     DECLARE DX INTEGER,
                 DY INTEGER,
                 DZ INTEGER;

229    3     DECLARE SIGNX BYTE,
                 SIGNY BYTE,
                 SIGNZ BYTE;

230    3     DECLARE XPLUS LITERALLY '50H', /* 'P','@','0',' ' */
                 YPLUS LITERALLY '40H',
                 XMINUS LITERALLY '30H',
                 YMINUS LITERALLY '20H';

231    3     DECLARE CHX BYTE ,
                 CHY BYTE ,
                 CHXY BYTE ;

232    3     DECLARE (M,N,F,MX,MY,DELX,DELY) INTEGER;
```

```
               /* Declarations for variables used in the circular inter
                     -polation */
233    3       DECLARE (II,JJ,XC,YC,X,Y,FX,FY,LX,LY,D1,D2,D3,D) INTEGER
                     -;

234    3       SIGNX=FALSE;
235    3       SIGNY=FALSE;
236    3       SIGNZ=FALSE;


237    3       IF INCREMENTAL=FALSE THEN DO;
239    4           DX=X2-X1;
240    4           DY=Y2-Y1;
241    4           DZ=Z2-Z1;
242    4           END;
243    3       ELSE DO;
244    4           DX=X2;
245    4           DY=Y2;
246    4           DZ=Z2;
247    4            END;

248    3       IF DX<0 THEN DO;
250    4           SIGNX=TRUE;
251    4           DX=-DX;
252    4           END;

253    3       IF DY<0 THEN DO;
255    4           SIGNY=TRUE;
256    4           DY=-DY;
257    4           END;

258    3       IF DZ<0 THEN DO;
260    4           SIGNZ=TRUE;
261    4           DZ=-DZ;
262    4           END;

263    3       CALL OUTCHAR('&');
264    3       IF SIGNX THEN CHX=XMINUS; ELSE CHX=XPLUS;
267    3       IF SIGNY THEN CHY=YMINUS; ELSE CHY=YPLUS;
270    3       IF SIGNZ THEN CALL OUTCHAR('H');
272    3       ELSE CALL OUTCHAR('I');
               /* PEN UP OR PEN DOWN */
273    3       CALL OUTCHAR('D');

274    3       M=DX+DY;
275    3       F=0;
276    3       MX=0;
```

```
277   3      MY=0;
278   3      N=0;
279   3      IF LINEAR THEN DO;
                 /*Linear Interpolation begins */

281   4          DELX=-DY;
282   4          DELY=DX;
283   4          IF (DX<>0) AND (DY<>0) THEN GOTO BOTH;
285   4          IF (DX=0) AND (DY=0) THEN GOTO DONE;
287   4          IF (DX=0) THEN CHXY=CHY; ELSE CHXY=CHX;
290   4          DO WHILE N<>M;
291   5          CALL OUTCHAR(CHXY);
292   5          CALL TIME(100);
293   5          N=N+1;
294   5          END;
295   4          GOTO DONE;
296   4          BOTH: IF (DX<DY) THEN GOTO YALSO;
298   4          XALSO: CALL OUTCHAR(CHX); CALL TIME(100);
300   4          N=N+1;
301   4          MX=MX+1;
302   4          IF N=M THEN GOTO DONE;
304   4          IF MX=DX THEN GOTO YALSO;
306   4          F=F+DELX;
307   4          IF F>=0 THEN GOTO XALSO;
309   4          YALSO: CALL OUTCHAR(CHY); CALL TIME(100);
311   4          N=N+1;
312   4          MY=MY+1;
313   4          IF N=M THEN GOTO DONE;
315   4          IF MY=DY THEN GOTO XALSO;
317   4          F=F+DELY;
318   4          IF F<=0 THEN GOTO YALSO;
320   4          GOTO XALSO;
321   4          DONE: /* Interpolation ends */

             END;

322   3      ELSE IF CIRCULAR THEN
323   3       DO; /* Circular interpolation begins */
324   4      II=CODENOW(11);
325   4      JJ=CODENOW(12);
326   4      XC=X1+II;
327   4      YC=Y1+JJ;
328   4      X=X1;
329   4      Y=Y1;
330   4      FX=X2;
331   4      FY=Y2;
332   4      IF (Y>YC) THEN DELX=1; ELSE DELX=-1;
```

```
335    4      IF (X>XC) THEN DELY=-1; ELSE DELY=1;
338    4      IF (X=XC) THEN DO;
340    5              IF (Y>YC) THEN DO;
342    6                      DELY=-1;
343    6                      DELX=1;
344    6                      END;
345    5              ELSE DO;
346    6                  DELY=1;
347    6                  DELX=-1;
348    6                   END;
349    5              END;
350    4      IF (Y=YC) THEN DO;
352    5                  IF (X>XC) THEN DO;
354    6                         DELX=-1;
355    6                         DELY=-1;
356    6                         END;
357    5                  ELSE DO;
358    6                      DELX=1;
359    6                      DELY=-1;
360    6                  END;
361    5              END;
362    4      IF CWISE=FALSE THEN DO;
364    5      IF (Y>YC) THEN DELX=-1;ELSE DELX=1;
367    5      IF (X>XC) THEN DELY=1;ELSE DELY=-1;
370    5      IF (X=XC) THEN DO;
372    6      IF (Y>YC) THEN DO;
374    7      DELY=-1;
375    7      DELX=-1;
376    7      END;
377    6      ELSE DO;
378    7      DELY=1;
379    7      DELX=1;
380    7      END;
381    6      END;
382    5      IF (Y=YC) THEN DO;
384    6      IF (X>XC) THEN DO;
386    7      DELX=-1;
387    7      DELY=1;
388    7      END;
389    6      ELSE DO;
390    7      DELX=1;
391    7      DELY=-1;
392    7      END;
393    6      END;
394    5      END;
395    4      LX=DELX*2*(X-XC)+1;
396    4      LY=DELY*2*(Y-YC)+1;
```

```
397    4        D=0;
398    4        DO WHILE ((X2=FX) OR (Y2=FY)
                OR (((X1-FX)*(X2-FX)<0) AND ((Y1-FY)*(Y2-FY)<0))
                OR (((X1-FX)*(X2-FX)>=0) AND ((Y1-FY)*(Y2-FY)<0))
                OR(((X1-FX)*(X2-FX)<0)AND((Y1-FY)*(Y2-FY)>=0)AND(Y2*FY>0
                      -))) ;
399    5        D1=(D+LX); IF D1<0 THEN D1=-D1;
402    5        D2=(D+LY); IF D2<0 THEN D2=-D2;
405    5        D3=(D+LX+LY); IF D3<0 THEN D3=-D3;
408    5        IF (D1>D2)AND (D2>D2) THEN GOTO XYINC;
410    5        IF (D1>D2)AND (D2<=D2) THEN GOTO YINC;
412    5        IF D1>D3 THEN GOTO XYINC;
414    5        XINC: D=D+LX;
415    5        LX=LX+2;
416    5        X=X+DELX;
417    5        CALL OUTCHAR(CHX);
418    5        GOTO CONTIN;

419    5        XYINC: D=D+LX+LY;
420    5        LX=LX+2;
421    5        LY=LY+2;
422    5        X=X+DELX;
423    5        Y=Y+DELY;
424    5        CALL OUTCHAR(CHX);
425    5        CALL OUTCHAR(CHY);
426    5        GOTO CONTIN;

427    5        YINC: D=D+LY;
428    5        LY=LY+2;
429    5        Y=Y+DELY;
430    5        CALL OUTCHAR(CHY);
431    5        CONTIN: X1=X2;
432    5        X2=X;
433    5        Y1=Y2;
434    5        Y2=Y;
435    5        END;

436    4        DO WHILE (X<>FX);
437    5        CALL OUTCHAR(CHX);
438    5        X=X+DELX;
439    5        END;

440    4        DO WHILE (Y<>FY) ;
441    5        CALL OUTCHAR(CHY);
442    5        Y=Y+DELY;
443    5        END;
```

```
 444   4        END;/* Circular interpolation ends */
 445   3       ELSE  DO;/* Point to point ,positioning */

 446   4           IF DX=DY THEN F=DX;
 448   4           ELSE IF DX>DY THEN DO;
 450   5                                CHXY=CHX;
 451   5                                F=DY;
 452   5                               END;
 453   4           ELSE DO;
 454   5                 F=DX;
 455   5                 CHXY=CHY;
 456   5                END;
 457   4           DO DELX=1 TO F;
 458   5           CALL OUTCHAR(CHX);
 459   5           CALL OUTCHAR(CHY);
 460   5           END;
 461   4           IF DX=DY THEN GOTO OVER;
 463   4           N=M-F-F;
 464   4           DO DELX=1 TO N;
 465   5           CALL OUTCHAR(CHXY);
 466   5           END;
 467   4       OVER:END;/* point to point positioning over */


 468   3           CALL OUTCHAR(CR);
 469   3           CALL OUTCHAR(LF);
 470   3           CALL OUTCHAR('&');  /* DELIMITER FOR PLOTTER DATA */
 471   3           DO I=2 TO 6;
 472   4           CODEPAST(I)=CODENOW(I);
 473   4           END;

 474   3       RET1:   END;  /* END OF PROCEDURE MOVE */

               /*ACTION STARTS HERE */
 475   2       CODE=CODENOW(1);
 476   2       SPEED=CODENOW(7);
 477   2       TOOL=CODENOW(10);
 478   2       IF SPEED<MAXSPEED THEN
 479   2       SPEED=(SPEED/MAXSPEED)*256;
 480   2       ELSE SPEED=MAXSPEED;
 481   2       I=LOW(UNSIGN(SPEED));
 482   2       P2CBUF=I;
 483   2       OUTPUT(P2C)=I;
 484   2       IF CODENOW(10)<>CODEPAST(10) THEN
 485   2           DO;
 486   3           CALL OUTSTRING(@TOOLMESS);
 487   3           CALL OUTINTEGER(TOOL);
```

```
488   3              CODENOW(10)=CODEPAST(10);
489   3              END;
490   2         ELSE IF CODE=00 THEN LINEAR=FALSE;/* point to point */
492   2         ELSE IF CODE=01 THEN LINEAR=TRUE;  /* linear interpolatio
                     -n */
494   2         ELSE IF (CODE=02)OR(CODE=03) THEN DO;/* circular interpo
                     -lation */
496   3                                        CIRCULAR=TRUE;
497   3                                        IF CODE=02 THEN CWISE=TRUE;
                     -ELSE CWISE=FALSE;
500   3                                        CALL MOVE;
501   3                                           END;
502   2         ELSE IF CODE=91 THEN INCREMENTAL=TRUE;/*incremental dime
                     -nsioning */
504   2         ELSE IF CODE=90 THEN INCREMENTAL=FALSE;/* absolute dimen
                     -sioning */
506   2         ELSE IF CODE=04  THEN CALL DELAY;  /* dwell */
508   2         ELSE IF (CODE=28) OR (CODE=29) THEN
509   2               DO;
510   4               DO I=2 TO 6;CODENOW(I)=0;
512   4               END;
513   3               CALL MOVE;
514   3               END;
515   2         ELSE IF CODE=53 THEN CALL MOVE;
517   2         ELSE IF (CODE<>90) AND (CODE<>91) AND (CODE<>4) AND (COD
                     -E<>28) AND (CODE<>29) AND (CODE<>53) THEN CALL NO
                     -TIMPLEM;

519   2         DO I=2 TO 6;
520   3         IF NOT (CODENOW(I)=CODEPAST(I)) THEN CALL MOVE;
522   3         END;
523   2         MISC=CODENOW(9);

524   2         IF MISC=00 THEN CALL HOLD;

526   2         ELSE IF MISC=02 THEN GOTO EXIT;/*END OF PROGRAM*/

528   2         ELSE IF MISC=03 THEN DO;
                  /* CW ROTATION */
530   3         P2ABUF=(P2ABUF) AND (5FH);
531   3         OUTPUT(P2A)=P2ABUF;
532   3         END;

533   2         ELSE IF MISC=04 THEN DO;
                     /* CCW ROTATION */
535   3              P2ABUF=(P2ABUF) OR (80H);
536   3              P2ABUF=(P2ABUF) AND (0DFH);
```

```
537   3             OUTPUT(P2A)=P2ABUF;
538   3             END;
539   2         ELSE IF MISC=05 THEN DO;
                    /* STOP SPINDLE */
541   3             P2ABUF=(P2ABUF) OR (0A0H);
542   3             OUTPUT(P2A)=P2ABUF;
543   3             END;

544   2         ELSE IF MISC=06  THEN DO;
                    /* TOOL CHANGE */
546   3             CALL OUTSTRING(@TOOLMESS);
547   3             CALL OUTINTEGER(TOOL);
548   3             END;

549   2         ELSE IF MISC=07 THEN DO;
                    /* COOLANT ON */
551   3             P1ABUF=(P1ABUF) OR (COOLNT);
552   3             OUTPUT(P1A)=P1ABUF;
553   3             END;

554   2         ELSE IF MISC=09 THEN DO;
                    /* COOLANT OFF */
556   3             P1ABUF=(P1ABUF) AND (0FEH);
557   3             OUTPUT(P1A)=P1ABUF;
558   3             END;

559   2         ELSE IF MISC=10 THEN DO;
                    /* CLAMP */
561   3         P2ABUF= P2ABUF OR (10H) ;
562   3         OUTPUT(P2A)=P2ABUF;
563   3         END;

564   2         ELSE IF MISC=11 THEN DO;
                    /* UNCLAMP */
566   3         P2ABUF=P2ABUF AND (0EFH);
567   3         OUTPUT(P2A)=P2ABUF;
568   3         END;
569   2         ELSE IF (MISC<>99) AND (MISC>11) THEN CALL NOTIMPLEM;
571   2          CODENOW(9)=99;/* MISC=99 */
572   2          CODEPAST(10)=CODENOW(10);
573   2         END; /* ACTION */


574   1         GET$NEW$CODE:PROCEDURE;

575   2         DIGIT$NO:PROCEDURE;
576   3         IF (CHAR>='0') AND (CHAR<='9') THEN II=CHAR-'0';
```

```
 578   3        ELSE II=0FFH;
 579   3        END;

 580   2        CODE$COORD:PROCEDURE;
 581   3        DECLARE MAX$DIGIT BYTE DATA (05H);
 582   3        J=0;K=I;
 584   3        M=0;
 585   3        LOOP1:INDEX=INDEX+1;
 586   3        CHAR=BUFFER(INDEX);
 587   3        IF CHAR=0 THEN GOTO CONT1;/* buffer end  */
 589   3        IF CHK$VALID=TRUE THEN GOTO CONT1;
 591   3        CALL DIGIT$NO; /*NOW II CONTAINS THE DIGIT VALUE */
 592   3        IF (II=0FFH) then
 593   3             if (CHAR<>'+')AND(CHAR<>'-') THEN GOTO CONT1;
 595   4        IF CHAR='-' THEN DO;M=1;GOTO LOOP1;END;
 600   3        IF CHAR='+' THEN GOTO LOOP1;
 602   3        T(J)=INT(II);
 603   3        J=J+1;
 604   3        IF J<(MAX$DIGIT) THEN GOTO LOOP1;
 606   3        INDEX=INDEX+1;
 607   3        CONT1: DO CASE J;
 608   4      . TEMP=0;
 609   4        TEMP=T(0);
 610   4        TEMP=T(0)*10+T(1);
 611   4        TEMP=T(0)*100+T(1)*10+T(2);
 612   4        TEMP=T(0)*1000+T(1)*100+T(2)*10+T(3);
 613   4        TEMP=T(0)*10000+T(1)*1000+T(2)*100+T(3)*10+T(4);
 614   4        END;
 615   3        IF (M=01) THEN
 616   3        TEMP=-TEMP;
 617   3        END;

 618   2        GET$NGSFTM:PROCEDURE;
 619   3        J=0;K=I;
 621   3        LOOP2:INDEX=INDEX+1;
 622   3        CHAR=BUFFER(INDEX);
 623   3        IF CHAR=0 THEN GOTO CONT2;
 625   3        IF CHK$VALID=TRUE THEN GOTO CONT2;
 627   3        CALL DIGIT$NO;
 628   3        IF (II=0FFH) THEN IF (CHAR<>'+') THEN GOTO CONT2;
 631   3        T(J)=INT(II);
 632   3        J=J+1;
 633   3        IF  (J<L) THEN GOTO LOOP2;
 635   3        INDEX=INDEX+1;
 636   3        CONT2: DO CASE J;
 637   4        TEMP=0;
 638   4        TEMP=T(0);
```

```
639   4        TEMP=T(0)*10+T(1);
640   4        TEMP=T(0)*100+T(1)*10+T(2);
641   4        END;
642   3        END;

643   2        CODE$3:PROCEDURE;/* FOR N,S,F */
644   3        L=3;
645   3        CALL GET$NGSFTM;
646   3        END;

647   2        CODE$2:PROCEDURE;/* FOR G,T,M */
648   3        L=2;
649   3        CALL GET$NGSFTM;
650   3        END;

651   2        INDEX=0;
652   2        LP:CHAR=BUFFER(INDEX);
653   2        IF CHAR=0 THEN GOTO EX;
655   2        IF CHK$VALID=TRUE THEN
656   2             DO CASE I;
657   3             CALL CODE$3;
658   3             CALL CODE$2;
659   3             CALL CODE$COORD;
660   3             CALL CODE$COORD;
661   3             CALL CODE$COORD;
662   3             CALL CODE$COORD;
663   3             CALL CODE$COORD;
664   3             CALL CODE$3;
665   3             CALL CODE$3;
666   3             CALL CODE$2;
667   3             CALL CODE$2;
668   3             CALL CODE$COORD;
669   3             CALL CODE$COORD;
670   3             END;   /*NOW TEMP CONTAINS THE INTEGER VALUE OF THE I
                        -'TH FUNCTION */
671   2         ELSE  DO;
672   3                CALL OUT$STRING(@ERR5);
673   3                CALL OUT$STRING(@WARN);
674   3                INDEX=INDEX+1;
675   3                GOTO LP;
676   3                END;
677   2            CODE$PAST(K)=CODE$NOW(K);
678   2            CODE$NOW(K)=TEMP;
679   2            GOTO LP;
680   2         EX: END;

681   1        INIT$PLOTTER:PROCEDURE;
```

```
682   2        DECLARE PLTCHR(*) BYTE DATA ('&H&',0);
683   2        CALL OUTSTRING(@PLTCHR);
684   2        END;

685   1        OUT$FILE:PROCEDURE;
686   2        DECLARE CH BYTE;
687   2        FILE$INDEX=0;
688   2        REPLP:CH=FILE$DATA(FILE$INDEX);
689   2        IF CH=0 THEN GOTO REPEX;
691   2        CALL OUTCHAR(CH);
692   2        IF CH=';' THEN DO;
694   3         CALL OUTCHAR(CR);
695   3         CALL OUTCHAR(LF);
696   3         END;
697   2        FILE$INDEX=FILE$INDEX+1;
698   2        GOTO REPLP;
699   2        REPEX:CALL OUTCHAR(CR);
700   2        CALL OUTCHAR(LF);
701   2        END;

702   1        START:CALL INITPORT;
703   1              CALL INITIALPORTS;
704   1              CALL INITIAL$8251;
705   1              CALL INITIAL$DATA;
706   1              CALL OUT$STRING(@SIGN$ON);
707   1              CALL OUTCRLF;
708   1         LOOP:CALL STORE$FILE;
709   1         CONTIN:CALL OUT$FILE;
710   1                CALL INITIAL$DATA;
711   1                CALL INIT$PLOTTER;
712   1        ACT$LOOP:CALL GET$DATA;
713   1        EXT$LOOP:CALL GET$NEW$CODE;
714   1                 CALL ACTION;
715   1                 GOTO ACT$LOOP;

716   1        EXIT:  CALL OUT$STRING(@MESS4);
717   1               CALL GETCHAR;
718   1               IF CHAR='N' THEN GOTO LOOP;
720   1               IF CHAR='C' THEN GOTO CONTIN;
722   1               GOTO EXIT;
723   1        CTLZ:  CALL OUTSTRING(@ABORT);
724   1               GOTO START;
725   1        END; /*MAIN PROGRAM*/
```

```
   DEFN  ADDR     SIZE  NAME, ATTRIBUTES, AND REFERENCES
   ----  ------   ----  ------------------------------


     4  002BH      16  ABORT. . . . . . .    BYTE ARRAY(16) DATA
                                              723
   202  0493H     785  ACTION . . . . . .    PROCEDURE STACK=001AH
                                              714
   712  0025H          ACTLOOP. . . . . .    LABEL      715
    13                 BKSPCE . . . . . .    LITERALLY '08H'
                                              150
    13                 BLANK. . . . . . .    LITERALLY '20H'
                                               67  161
   128  0506H       1  BLK. . . . . . . .    BYTE IN PROC (STOREFILE)
                                              131*  167*  167
    17 FHFFF0H      1  BOOT1. . . . . . .    BYTE ARRAY(1) AT INITIAL A
                                              -BSOLUTE
    17 FHFFF1H      2  BOOT2. . . . . . .    WORD ARRAY(1) AT INITIAL A
                                              -BSOLUTE
    17 FHFFF3H      2  BOOT3. . . . . . .    WORD ARRAY(1) AT INITIAL A
                                              -BSOLUTE
   296  09B4H          BOTH . . . . . . .    LABEL IN PROC (MOVE)
                                              284
    14  0479H     128  BUFFER . . . . . .    BYTE ARRAY(128)
                                              100*  103*  110*  586
                                              622  652
   686  050DH       1  CH . . . . . . . .    BYTE IN PROC (OUTFILE)
                                              688*  689  691  692
    56  0004H       1  CH . . . . . . . .    BYTE IN PROC (OUTCHAR) PAR
                                              -AMETER AUTOMATIC
                                               56   61
     9  0090H       1  CHAR . . . . . . .    BYTE           66*   67
                                               68   69   89  135
                                              137  140  142  145
                                              147  150  156  161
                                              163  165  170  188*
                                              189  190  194*  195
                                              199  576  577  586*
                                              587  593  595  600
                                              622*  623  629  652*
                                              653  718  720
    50  0112H      28  CHARRDY. . . . . .    PROCEDURE BYTE STACK=0002H
                                               57  192
   187  043EH      85  CHKCTLCHAR . . . .    PROCEDURE STACK=0006H
                                               58
    87  01CFH      54  CHKVALID . . . . .    PROCEDURE BYTE STACK=0002H
```

```
                                              589  625  655
  231  050AH      1  CHX. . . . . . . . .   BYTE IN PROC (MOVE)
                                              265*  266*  289  298
                                              417  424  437  450
                                              458
  231  050CH      1  CHXY . . . . . . . .   BYTE IN PROC (MOVE)
                                              288*  289*  291  450*
                                              455*  465
  231  050BH      1  CHY. . . . . . . . .   BYTE IN PROC (MOVE)
                                              268*  269*  288  309
                                              425  430  441  455
                                              459
   20  04FFH      1  CIRCULAR . . . . . .   BYTE        117*  322
                                              496*
   12                CMND8251 . . . . . .   LITERALLY '025H'
                                              47
  203  004EH      2  CODE . . . . . . . .   INTEGER IN PROC (ACTION)
                                              475*  490  492  494
                                              497  502  504  506
                                              508  515  517
  647  12B2H     13  CODE2. . . . . . . .   PROCEDURE IN PROC (GETNEWC
                                              -ODE) STACK=000AH
                                              658  666  667
  643  12A5H     13  CODE3. . . . . . . .   PROCEDURE IN PROC (GETNEWC
                                              -ODE) STACK=000AH
                                              657  664  665
  580  1039H    392  CODECOORD. . . . . .   PROCEDURE IN PROC (GETNEWC
                                              -ODE) STACK=0006H
                                              659  660  661  662
                                              663  668  669
   14  0026H     26  CODENOW. . . . . . .   INTEGER ARRAY(13)
                                              122*  124*  125*  221
                                              224*  239  240  241
                                              244  245  246  324
                                              325  330  331  398
                                              431  432*  433  434*
                                              472  475  476  477
                                              484  488*  511*  520
                                              523  571*  572  677
                                              678*
   14  000CH     26  CODEPAST . . . . . .   INTEGER ARRAY(13)
                                              121*  224  239  240
                                              241  326  327  328
                                              329  398  431*  433*
                                              472*  484  488  520
                                              572*  677*
```

```
607   10ECH        CONT1. . . . . . . .    LABEL IN PROC (CODECOORD)
                                            588  590  594
636   1245H        CONT2. . . . . . . .    LABEL IN PROC (GETNGSFTM)
                                            624  626  630
431   0DEFH        CONTIN . . . . . . .    LABEL IN PROC (MOVE)
                                            418  426
709   001CH        CONTIN . . . . . . .    LABEL        721
 21                COOLNT . . . . . . .    LITERALLY '01H'
                                            551
  3                CR . . . . . . . . .    LITERALLY '0DH'
                                              4    5   69   73
                                            161  468  694  699
723   0062H        CTLZ . . . . . . . .    LABEL        196  200
 20   0500H    1   CWISE. . . . . . . .    BYTE         118* 362
                                            498* 499*
233   0086H    2   D. . . . . . . . . .    INTEGER IN PROC (MOVE)
                                            397* 399  402  405
                                            414* 414  419* 419
                                            427* 427
233   0080H    2   D1 . . . . . . . . .    INTEGER IN PROC (MOVE)
                                            399* 400  401* 401
                                            408  410  412
233   0082H    2   D2 . . . . . . . . .    INTEGER IN PROC (MOVE)
                                            402* 403  404* 404
                                            408  410
233   0084H    2   D3 . . . . . . . . .    INTEGER IN PROC (MOVE)
                                            405* 406  407* 407
                                            412
 12                DATA8251 . . . . . .    LITERALLY '0FFF0H'
                                             61   66  188  194
219   07E2H   55   DELAY. . . . . . . .    PROCEDURE IN PROC (ACTION)
                                            - STACK=0002H
                                            507
 13                DELETE . . . . . . .    LITERALLY '010H'
                                            150
232   0068H    2   DELX . . . . . . . .    INTEGER IN PROC (MOVE)
                                            281* 306  333* 334*
                                            343* 347* 354* 358*
                                            365* 366* 375* 379*
                                            386* 390* 395  416
                                            422  438  457* 457
                                            460  464* 464  466
232   006AH    2   DELY . . . . . . . .    INTEGER IN PROC (MOVE)
                                            282* 317  336* 337*
                                            342* 346* 355* 359*
```

```
                                             368*  369*  374*  378*
                                             387*  391*  396   423
                                             429   442
  8   017DH     10   DIGIT. . . . . . .     BYTE ARRAY(10) DATA
575   1011H     40   DIGITNO. . . . . .     PROCEDURE IN PROC (GETNEWC
                                             -ODE) STACK=0002H
                                             591   627
321   0A6BH          DONE . . . . . . .     LABEL IN PROC (MOVE)
                                             286   295   303   314
 12                  DSRDY. . . . . . .     LITERALLY '080H'
228   0058H      2   DX . . . . . . . .     INTEGER IN PROC (MOVE)
                                             239*  244*  248   251*
                                             251   274   282   283
                                             285   287   296   304
                                             446   447   448   454
                                             461
228   005AH      2   DY . . . . . . . .     INTEGER IN PROC (MOVE)
                                             240*  245*  253   256*
                                             256   274   281   283
                                             285   296   315   446
                                             448   451   461
228   005CH      2   DZ . . . . . . . .     INTEGER IN PROC (MOVE)
                                             241*  246*  258   261*
                                             261
 13                  EOB. . . . . . . .     LITERALLY '3BH'
  4   003BH     25   ERR1 . . . . . . .     BYTE ARRAY(25) DATA
                                             217
  4   0054H     24   ERR2 . . . . . . .     BYTE ARRAY(24) DATA
  6   013AH     27   ERR5 . . . . . . .     BYTE ARRAY(27) DATA
                                             672
680   100FH          EX . . . . . . . .     LABEL IN PROC (GETNEWCODE)
                                             654
716   0036H          EXIT . . . . . . .     LABEL         98    527
                                             722
713   0028H          EXTLOOP. . . . . .     LABEL        105
232   0062H      2   F. . . . . . . . .     INTEGER IN PROC (MOVE)
                                             275*  306*  306   307
                                             317*  317   318   447*
                                             451*  454*  457   463
  5                  FALSE. . . . . . .     LITERALLY '000H'
                                              52    92   117   234
                                             235   236   237   362
                                             491   499   505
 11   0091H   1000   FILEDATA . . . . .     BYTE ARRAY(1000)
                                              97    99   100   101
```

```
                                          137*  142*  147*  158*
                                          165*  182*  688
 10  000AH     2  FILEINDEX. . . . .     WORD          97    99
                                          100   101   108*  108
                                          111*  111   114*  129*
                                          138*  138   143*  143
                                          148*  148   152*  152
                                          153   154*  166   168*
                                          168   172*  178   687*
                                          688   697*  697
  9  008FH     1  FL . . . . . . . .     BYTE          97   104*
                                          119*
233  0078H     2  FX . . . . . . . .     INTEGER IN PROC (MOVE)
                                          330*  398   436
233  007AH     2  FY . . . . . . . .     INTEGER IN PROC (MOVE)
                                          331*  398   440
 63  0158H    56  GETCHAR. . . . . .     PROCEDURE STACK=000CH
                                          134   717
 94  0205H   109  GETDATA. . . . . .     PROCEDURE STACK=0002H
                                          712
574  0F3CH   213  GETNEWCODE . . . .     PROCEDURE STACK=0016H
                                          713
618  11C1H   228  GETNGSFTM. . . . .     PROCEDURE IN PROC (GETNEWC
                                          -ODE) STACK=0006H
                                          645   649
210  07BAH    28  HOLD . . . . . . .     PROCEDURE IN PROC (ACTION)
                                          - STACK=0002H
                                          525
  9  0088H     1  I. . . . . . . . .     BYTE          88*   88
                                          89    91    120*  120
                                          123   211*  212*  212
                                          213   471*  471   472
                                          473   481*  482   483
                                          510*  510   512   519*
                                          519   520   522   583
                                          620   656
233  006CH     2  II . . . . . . . .     INTEGER IN PROC (MOVE)
                                          324*  326
  9  008EH     1  II . . . . . . . .     BYTE          577*  578*
                                          592   602   628   631
 20  04FDH     1  INCREMENTAL. . . .     BYTE          115*  237
                                          503*  505*
 95  0505H     1  IND. . . . . . . .     BYTE IN PROC (GETDATA)
                                          96*   107*  107
 79  0504H     1  IND. . . . . . . .     BYTE IN PROC (OUTSTRING)
                                          80*   81    82    83*
```

```
                                                83
  15   04F9H     1   INDEX. . . . . . . .    BYTE         585*  585
                                              586  606*  606  621*
                                              621  622  635*  635
                                              651*  652  674*  674
  38   00B5H    93   INITIAL8251. . . .      PROCEDURE STACK=0002H
                                              704
 113   0272H    88   INITIALDATA. . . .      PROCEDURE STACK=0002H
                                              705  710
  31   0096H    31   INITIALPORTS . . .      PROCEDURE STACK=0002H
                                              703
 681   12BFH    12   INITPLOTTER. . . .      PROCEDURE STACK=0016H
                                              711
  23   0071H    37   INITPORT . . . . . .    PROCEDURE STACK=0002H
                                              702
                     INPUT. . . . . . . .    BUILTIN            51
                                               59   64   66  188
                                              194  211
                     INT. . . . . . . . .    BUILTIN           602
                                              631
   2   0000H    10   INTVECTOR. . . . .      POINTER ARRAY(5)
   9   0089H     1   J. . . . . . . . . .    BYTE         582*  603*
                                              603  604  607  619*
                                              632*  632  633  636
 233   006EH     2   JJ . . . . . . . . .    INTEGER IN PROC (MOVE)
                                              325*  327
   9   008AH     1   K. . . . . . . . . .    BYTE         583*  620*
                                              677
   9   008BH     1   L. . . . . . . . . .    BYTE         633  644*
                                              648*
                     LAST . . . . . . . .    BUILTIN            88

   3                 LF . . . . . . . . .    LITERALLY 'OAH'
                                                4    5   74  161
                                              469  695  700
 180   0187H    35   LIMITERR . . . . . .    BYTE ARRAY(35) IN PROC (ST
                                              -OREFILE) DATA
                                              181
  20   04FEH     1   LINEAR . . . . . . .    BYTE         116*  279
                                              491*  493*
 708   0019H         LOOP . . . . . . . .    LABEL        719
 134   02EAH         LOOP . . . . . . . .    LABEL IN PROC (STOREFILE)
                                              162  185
 585   104FH         LOOP1. . . . . . . .    LABEL IN PROC (CODECOORD)
                                              598  601  605
 621   11D1H         LOOP2. . . . . . . .    LABEL IN PROC (GETNGSFTM)
```

```
                                                  83
  15   04F9H      1   INDEX. . . . . . . .      BYTE         585*  585
                                                  586  606*  606  621*
                                                  621  622  635*  635
                                                  651*  652  674*  674
  38   00B5H     93   INITIAL8251. . . .        PROCEDURE STACK=0002H
                                                  704
 113   0272H     88   INITIALDATA. . . .        PROCEDURE STACK=0002H
                                                  705  710
  31   0096H     31   INITIALPORTS . . .        PROCEDURE STACK=0002H
                                                  703
 681   12BFH     12   INITPLOTTER. . . .        PROCEDURE STACK=0016H
                                                  711
  23   0071H     37   INITPORT . . . . .        PROCEDURE STACK=0002H
                                                  702
                      INPUT. . . . . . .        BUILTIN             51
                                                   59   64   66  188
                                                  194  211
                      INT. . . . . . . .        BUILTIN            602
                                                  631
   2   0000H     10   INTVECTOR. . . . .        POINTER ARRAY(5)
   9   0089H      1   J. . . . . . . . .        BYTE         582*  603*
                                                  603  604  607  619*
                                                  632*  632  633  636
 233   006EH      2   JJ . . . . . . . .        INTEGER IN PROC (MOVE)
                                                  325*  327
   9   008AH      1   K. . . . . . . . .        BYTE         583*  620*
                                                  677
   9   008BH      1   L. . . . . . . . .        BYTE         633  644*
                                                  648*
                      LAST . . . . . . .        BUILTIN             88

   3                  LF . . . . . . . .        LITERALLY 'OAH'
                                                     4    5   74  161
                                                  469  695  700
 180   0187H     35   LIMITERR . . . . .        BYTE ARRAY(35) IN PROC (ST
                                                  -OREFILE) DATA
                                                  181
  20   04FEH      1   LINEAR . . . . . .        BYTE         116*  279
                                                  491*  493*
 708   0019H          LOOP . . . . . . .        LABEL        719
 134   02EAH          LOOP . . . . . . .        LABEL IN PROC (STOREFILE)
                                                  162  185
 585   104FH          LOOP1. . . . . . .        LABEL IN PROC (CODECOORD)
                                                  598  601  605
 621   11D1H          LOOP2. . . . . . .        LABEL IN PROC (GETNGSFTM)
```

```
                                          634
                     LOW. . . . . . . . . BUILTIN          207
                                          481
                                          LABEL IN PROC (GETNEWCODE)
 652  0F44H          LP . . . . . . . . .  675  679
                                          WORD IN PROC (STOREFILE)
 128  004CH      2   LSTBLK . . . . . . .  130*  166*  172
                                          INTEGER IN PROC (MOVE)
 233  007CH      2   LX . . . . . . . . .  395*  399  405  414
                                           415*  415  419  420*
                                           420
                                          INTEGER IN PROC (MOVE)
 233  007EH      2   LY . . . . . . . . .  396*  402  405  419
                                           421*  421  427  428*
                                           428
                                          INTEGER IN PROC (MOVE)
 232  005EH      2   M. . . . . . . . . .  274*  290  302  313
                                           463
                                          BYTE           584*  597*
   9  008CH      1   M. . . . . . . . . .  615
                                          BYTE IN PROC (CODECOORD) D
 581  01CBH      1   MAXDIGIT . . . . . .  -ATA        604
                                          INTEGER IN PROC (ACTION) D
 204  0000H      2   MAXSPEED . . . . . .  -ATA        478  479
                                           480
                                          BYTE ARRAY(52) DATA
   5  006CH     52   MESS1. . . . . . . .  132
                                          BYTE ARRAY(52) DATA
   5  00A0H     52   MESS2. . . . . . . .  133
                                          BYTE ARRAY(51) DATA
   5  00D4H     51   MESS3. . . . . . . .  176
                                          BYTE ARRAY(51) DATA
   5  0107H     51   MESS4. . . . . . . .  716
                                          INTEGER IN PROC (ACTION)
 203  0050H      2   MISC . . . . . . . .  523*  524  526  528
                                           533  539  544  549
                                           554  559  564  569

                                          LITERALLY 'OCFH'
  12                 MODE8251 . . . . . .  45
                                          PROCEDURE IN PROC (ACTION)
 226  0819H   1827   MOVE . . . . . . . .  - STACK=000CH
                                           500  513  516  521
                                          INTEGER IN PROC (MOVE)
 232  0064H      2   MX . . . . . . . . .  276*  301*  301  304
                                          INTEGER IN PROC (MOVE)
 232  0066H      2   MY . . . . . . . . .
```

```
                                                  277*  312*  312  315
  232  0060H      2  N. . . . . . . . . . .      INTEGER IN PROC (MOVE)
                                                  278*  290   293*  293
                                                  300*  300   302   311*
                                                  311   313   463*  464
    9  008DH      1  N. . . . . . . . . . .      BYTE
  220  0056H      2  N. . . . . . . . . . .      WORD IN PROC (DELAY)
                                                  221*  221   223
       0000H    113  NCM. . . . . . . . . .      PROCEDURE STACK=001CH
  216  07D6H     12  NOTIMPLEM. . . . . . .      PROCEDURE IN PROC (ACTION)
                                                  - STACK=0016H
                                                  518   570
  206  0004H      2  NUMBER . . . . . . . .      INTEGER IN PROC (OUTINTEGE
                                                  -R) PARAMETER AUTOMATIC
                                                  206   207
   55  012EH     42  OUTCHAR. . . . . . . .      PROCEDURE STACK=0008H
                                                   68    73    74    82
                                                  263   271   272   273
                                                  291   298   309   417
                                                  424   425   430   437
                                                  441   458   459   465
                                                  468   469   470   691
                                                  694   695   699   700
   72  0190H     17  OUTCRLF. . . . . . . .      PROCEDURE STACK=000CH
                                                   70    85   173   707
  685  12CBH     81  OUTFILE. . . . . . . .      PROCEDURE STACK=000CH
                                                  709
  205  07A4H     22  OUTINTEGER . . . . . .      PROCEDURE IN PROC (ACTION)
                                                  - STACK=0004H
                                                  487   547
                     OUTPUT . . . . . . . .      BUILTIN               24*
                                                   28*   29*   32*   34*
                                                   36*   39*   41*   43*
                                                   45*   47*   61*  208*
                                                  483*  531*  537*  542*
                                                  552*  557*  562*  567*
   76  01A1H     46  OUTSTRING. . . . . . .      PROCEDURE STACK=0012H
                                                  132   133   176   181
                                                  217   486   546   672
                                                  673   683   706   716
                                                  723
  467  0F00H         OVER . . . . . . . . .      LABEL IN PROC (MOVE)
                                                  462
   21                P1A. . . . . . . . . .      LITERALLY 'OFFF9H'
                                                   28   552   557
   22  0501H      1  P1ABUF . . . . . . . .      BYTE           25*   28
```

```
                                        551*  551   552   556*
                                        556   557
21                     P1B. . . . . . . LITERALLY 'OFFFBH'
                                        211
22   0502H       1     P1BBUF . . . . . BYTE          26*
21                     P1C. . . . . . . LITERALLY 'OFFFDH'
                                        29   208
22   0503H       1     P1CBUF . . . . . BYTE          27*   29
                                        207*  208
21                     P1S. . . . . . . LITERALLY 'OFFFFH'
                                        24
19                     P2A. . . . . . . LITERALLY 'OFFF8H'
                                        34   531   537   542
                                        562  567
18   04FAH       1     P2ABUF . . . . . BYTE          33*   34
                                        530*  530   531   535*
                                        535   536*  536   537
                                        541*  541   542   561*
                                        561   562   566*  566
                                        567
19                     P2B. . . . . . . LITERALLY 'OFFFAH'
18   04FBH       1     P2BBUF . . . . . BYTE
19                     P2C. . . . . . . LITERALLY 'OFFFCH'
                                        36   483
18   04FCH       1     P2CBUF . . . . . BYTE          35*   36
                                        482*
19                     P2S. . . . . . . LITERALLY 'OFFFEH'
                                        32
682  01CCH       4     PLTCHR . . . . . BYTE ARRAY(4) IN PROC (INI
                                        -TPLOTTER) DATA
                                        683
77   0004H       2     PTR. . . . . . . POINTER IN PROC (OUTSTRING
                                        -) PARAMETER AUTOMATIC
                                        77    81    82
211  07BDH             REP. . . . . . . LABEL IN PROC (HOLD)
                                        214
699  130EH             REPEX. . . . . . LABEL IN PROC (OUTFILE)
                                        690
688  12D4H             REPLP. . . . . . LABEL IN PROC (OUTFILE)
                                        698
12                     RESET8251. . . . LITERALLY '065H'
                                        43
474  0F3AH             RET1 . . . . . . LABEL IN PROC (MOVE)
12                     RXRDY. . . . . . LITERALLY '02H'
                                        51    64
13                     SCRNBLNK . . . . LITERALLY 'OCH'
```

```
    4  0002H    41  SIGNON . . . . . . .    BYTE ARRAY(41) DATA
                                             706
  229  0507H     1  SIGNX. . . . . . . .    BYTE IN PROC (MOVE)
                                             234*  250*  264
  229  0508H     1  SIGNY. . . . . . . .    BYTE IN PROC (MOVE)
                                             235*  255*  267
  229  0509H     1  SIGNZ. . . . . . . .    BYTE IN PROC (MOVE)
                                             236*  260*  270
  203  0052H     2  SPEED. . . . . . . .    INTEGER IN PROC (ACTION)
                                             476*  478   479*  479
                                             480*  481
  702  0003H        START. . . . . . . .    LABEL        724
   17               STARTADDR. . . . . .    LITERALLY '0000H'
                                               17
   12               STAT8251 . . . . . .    LITERALLY '0FFF2H'
                                               39    41    43    45
                                               47    51    59    64
  127  02CAH   372  STOREFILE. . . . . .    PROCEDURE STACK=0016H
                                             708
   78  0000H     1  STR. . . . . . . . .    BYTE BASED(PTR) ARRAY(1) I
                                             -N PROC (OUTSTRING)
                                               81    82
   14  0040H    10  T. . . . . . . . . .    INTEGER ARRAY(5) INITIAL
                                             602*  609   610   611
                                             612   613   631*  638
                                             639   640
   16  004AH     2  TEMP . . . . . . . .    INTEGER                608*
                                             609*  610*  611*  612*
                                             613*  616*  616   637*
                                             638*  639*  640*  678
                    TIME . . . . . . . .    BUILTIN                 40
                                               42    44    46    48
                                              222   292   299   310
  203  0054H     2  TOOL . . . . . . . .    INTEGER IN PROC (ACTION)
                                             477*  487   547
  203  01AAH    33  TOOLMESS . . . . . .    BYTE ARRAY(33) IN PROC (AC
                                             -TION) DATA          486
                                             546
    5               TRUE . . . . . . . .    LITERALLY '0FFH'
                                               53    90   115   116
                                              118   250   255   260
                                              493   496   498   503
                                              589   625   655
   12               TXRDY. . . . . . . .    LITERALLY '01H'
                                               59
                    UNSIGN . . . . . . .    BUILTIN                207
```

```
                                                  481
   7  0170H    13  VALIDFN. . . . . .          BYTE ARRAY(13) DATA
                                                   88   89
   6  0155H    27  WARN . . . . . . . .        BYTE ARRAY(27) DATA
                                                  673
 233  0074H     2  X. . . . . . . . . .        INTEGER IN PROC (MOVE)
                                                  328*  335  338  352
                                                  367  370  384  395
                                                  416*  416  422*  422
                                                  432  436  438*  438
 227               X1 . . . . . . . . .        LITERALLY 'CODEPAST(2)'  I
                                                 -N PROC (MOVE)
                                                  239  326  328  398
                                                  431
 227               X2 . . . . . . . . .        LITERALLY 'CODENOW(2)'  IN
                                                 - PROC (MOVE)
                                                  239  244  330  398
                                                  431
 298  09C4H        XALSO. . . . . . . .        LABEL IN PROC (MOVE)
                                                  308  316  320
 233  0070H     2  XC . . . . . . . . .        INTEGER IN PROC (MOVE)
                                                  326*  335  338  352
                                                  367  370  384  395
 414  0D76H        XINC . . . . . . . .        LABEL IN PROC (MOVE)
 230               XMINUS . . . . . . .        LITERALLY '30H'  IN PROC
                                                 -MOVE)          265
 230               XPLUS. . . . . . . .        LITERALLY '50H'  IN PROC
                                                 -MOVE)          266
 419  0D96H        XYINC. . . . . . . .        LABEL IN PROC (MOVE)
                                                  409  413
 233  0076H     2  Y. . . . . . . . . .        INTEGER IN PROC (MOVE)
                                                  329*  332  340  350
                                                  364  372  382  396
                                                  423*  423  429*  429
                                                  434  440  442*  442
 227               Y1 . . . . . . . . .        LITERALLY 'CODEPAST(3)'
                                                 -N PROC (MOVE)
                                                  240  327  329  398
                                                  432
 227               Y2 . . . . . . . . .        LITERALLY 'CODENOW(3)'
                                                 - PROC (MOVE)
                                                  240  245  331  398
                                                  433
 309  0A16H        YALSO. . . . . . . .        LABEL IN PROC (MOVE)
                                                  297  305  319
 233  0072H     2  YC . . . . . . . . .        INTEGER IN PROC (MOVE)
```

```
                                                 327*  332  340  350
                                                 364  372  382  396
   427  ODD2H          YINC . . . . . . .       LABEL IN PROC (MOVE)
                                                 411
   230                 YMINUS . . . . . .       LITERALLY '20H'  IN PROC
                                                 -MOVE)           268
   230                 YPLUS. . . . . . .       LITERALLY '40H'  IN PROC
                                                 -MOVE)           269
   227                 Z1 . . . . . . . .       LITERALLY 'CODEPAST(4)'
                                                 -N PROC (MOVE)
                                                 241
   227                 Z2 . . . . . . . .       LITERALLY 'CODENOW(4)'  :
                                                 - PROC (MOVE)
                                                 241  246


MODULE INFORMATION:

     CODE AREA SIZE     = 131CH   4892D
     CONSTANT AREA SIZE = 01D0H    464D
     VARIABLE AREA SIZE = 050EH   1294D
     MAXIMUM STACK SIZE = 001CH     28D
     798 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

REFERENCES

1. McGlynn, Daniel, R.,
   'Modern microprocessor system design : sixteen bit and bit slice architecture',
   John Wiley and Sons, Inc., New York, 1980.
2. Childs, James J.,
   'Numerical control part programming',
   Industrial Press Inc., New York.
3. Roberts, Arthur D., et al.,
   'Programming for numerical control machines',
   McGraw Hill Book Company, New York, 1978.
4. Intel Corporation,
   'Component Data Catalog-1981',
   Intel Corporation, Santa Clara, CA, 1981.
5. Intel Corporation,
   'SDK-86: MCS-86 System Design Kit User's Guide',
   Intel Corporation, Santa Clara, CA, 1978.
6. Bianculli, Anthony J.,
   'Stepper motors: Application and Selection',
   IEEE Spectrum, Dec. 1970.
7. Electronics Industries Association,
   'EIA Standard RS-244A',
   Electronics Industries Association, USA, 1967.

8. Bergren C.,
'A Simple Algorithm for Circular Interpolation',
Control Engg., Sep. 1971, pp 57-59.

9. Yoram Koren,
'Computer Control of Manufacturing Systems',
McGraw Hill Book Co., New York, 1983.

10. Yoram Koren,
'Interpolation for a Computer Numerical Control System',
IEEE Trans. on Computers, Vol. C-25, No.1, Jan, 1976.

11. Yoram Koren, Masory O.,
'Reference Pulse Circular Interpolators for CNC Systems',
ASME Journal of Engg. for Industry, Vol. 103, No.1,
February 1981, pp 131-136.

12. Mayarov F.V.,
'Electronic Digital Integrating Computers - Digital
Differential Analyzers',
Illife Book, London, England, 1964.

13. --------------- and Yoram Koren,
'Reference Word Circular Interpolators for CNC Systems',
Trans. ASME Journal of Engg. for Industry, vol. 104,
Nov. 1982.